UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO DE FIN DE GRADO

IMPLEMENTACIÓN DE RUNNER ROOT ALGORITHM PARA TÉCNICAS DE CONTROL BILINEAL

Junio 2017

AUTOR:
Marcos Marín Rodríguez

TUTOR:
Fernando Martín Monar

Agradecimientos

En primer lugar cabe mencionar la labor de mi tutor Fernando Martín por animarme a emprender este proyecto, por guiarme y ayudarme en su elaboración.

Además, agradecer a todos los compañeros que he tenido a lo largo de mi paso por la universidad, desde aquellos que comenzaron conmigo este camino con el grupo bilingüe de industriales, hasta con los que he coincidido en este último curso. Mención especial para Víctor Alonso Gordón y David Aguiar Cuesta, con los que he compartido desde el inicio hasta el final este camino, pasando por una de mis mejores experiencias en el extranjero con el programa de movilidad no europea de la universidad en Corea del Sur. También me gustaría mencionar a Ximena de Diego, una excelente compañera y por supuesto mi mejor apoyo durante este tiempo.

Por último, agradecer el apoyo de toda mi familia por su confianza en mí y en mi capacidad de sacar todo adelante, especialmente mi hermano a quien animo a seguir con su camino particular en su carrera.

Resumen

Los algoritmos de optimización metaheurísticos son capaces de resolver muchos problemas complejos de ingeniería cuyas soluciones no se podrían obtener de manera efectiva usando algoritmos de optimización clásicos. Un alto porcentaje de algoritmos metaheurísticos imita el comportamiento de seres vivos o fenómenos físicos para cumplir una optimización. Este tipo de patrón de optimización siempre es iterativo y tiene un tipo de memoria para recordar las mejores soluciones de las iteraciones anteriores y para proporcionar los mejores "agentes" o representantes de la colonia con una mayor probabilidad de sobrevivir y reproducirse.

El algoritmo RRA ("Runner Root Algorithm") es un algoritmo genético, una variedad de los algoritmos metaheurísticos, para resolver problemas de optimización. El algoritmo propuesto se inspira en las plantas como la planta de la fresa o la planta de araña las cuales se propagan a través de sus tallos en la naturaleza, a la vez que simultáneamente desarrollan raíces y pelos radicales para una búsqueda local de fuentes de agua y minerales

El controlador BPID es el tipo de controlador propuesto en este trabajo para resolver los diferentes problemas de control. Este tipo de controlador se puede aplicar a sistemas no lineales, y está formado por un PID convencional y un compensador bilineal. La función del algoritmo RRA es encontrar los valores óptimos de las ganancias del controlador que minimicen la función coste.

Este proyecto se centra en la implementación del algoritmo RRA para resolver problemas de control mediante el controlador BPID. El principal problema de este trabajo es el control del funcionamiento de un motor de corriente continua, tanto de su posición en un primer caso, como de su velocidad posteriormente. Para ello se implementa un controlador BPID optimizado mediante el algoritmo RRA desarrollado y ajustado previamente. De esta manera, los mejores resultados son aquellos que minimicen el error del controlador.

Para aplicar el algoritmo a estas técnicas de control se parte de otro problema más sencillo que permita asentar las bases del funcionamiento del propio algoritmo. En este caso, se resuelve el problema de la distancia mínima al punto que permite comprender de manera gráfica dicho funcionamiento.

Analizando los resultados obtenidos, se puede concluir que el algoritmo RRA es una técnica alternativa válida para la resolución de problemas de control.

Abstract

Metaheuristic optimization algorithms are able to solve many complex engineering problems whose solutions could not be effectively obtained using classical optimization algorithms. In fact, a high percentage of metaheuristic algorithms imitate the behavior of living beings or physical phenomena to fulfill an optimization. This type of optimization pattern is always iterative and has a type of memory to remember the best solutions of previous iterations and to provide the best "agents" or representatives of the colony with a greater probability of surviving and reproducing.

The RRA ("Runner Root Algorithm") is a genetic algorithm, a variety of metaheuristic algorithms, to solve optimization problems. The proposed algorithm is inspired by plants such as the strawberry plant or the spider plant which propagate through their runners in nature, once they simultaneously develop roots and root hairs for a local search of water sources and minerals.

The BPID controller is the type of controller proposed in this work to solve different control problems. This type of controller can be applied to non-linear systems, and consists of a conventional PID and a bilinear compensator. The function of the RRA algorithm is to find the optimal values of the controller gains that minimize the cost function.

This project focuses on the implementation of RRA algorithm to solve control problems using the BPID controller. The main problem of this work is the control of the operation of a DC motor, both its position in a first case and its speed afterwards. For this purpose a BPID controller is implemented optimized by the RRA algorithm developed and adjusted previously. In this way, the best results are those which minimize the error of the controller.

In order to apply the algorithm to these control techniques, it is necessary its application to a simpler problem that allows to comprehend the bases of the operation of the algorithm. In this case, the problem of the minimum distance to the point is solved, which allows to graphically understand its performance.

Analyzing the obtained results, it is possible to conclude that the RRA algorithm is a valid alternative technique for the resolution of control problems.

Índice

Capítulo 1. Introducción

Durante este capítulo, se realiza un estudio preliminar en el que se enfoca el proyecto, se evalúa su entorno de aplicación y se presenta el algoritmo utilizado a lo largo del proyecto. Además, recoge la metodología, estructura y la planificación del proyecto.

La estructura de este capítulo se divide en seis secciones: En la primera de ellas se comenta el enfoque del proyecto. En la segunda Sección se resume el entorno de aplicación, así como del impacto socio-económico del proyecto. En la siguiente Sección se introduce el algoritmo RRA y la cuarta Sección describe la metodología utilizada en el documento. La sexta Sección se corresponde con la planificación del proyecto.

1.1 Enfoque y motivación

Durante las últimas cuatro décadas, la naturaleza ha sido una fuente de inspiración para el desarrollo de nuevos algoritmos de optimización para resolver complejos problemas de ingeniería.

Sin embargo, el primer avance general y desarrollado de este tipo de algoritmos es el algoritmo genético (AG) desarrollado por Holland [1]. A partir de entonces, se han implementado estas técnicas para el ajuste y optimización de nuevos métodos de control.

Dada la complejidad de los nuevos sistemas a controlar (sistemas no lineales que son incontrolables mediante las técnicas control convencionales) se implementa un nuevo método de control bilineal, el controlador BPID.

Este proyecto está enfocado al uso del RRA para optimizar los parámetros de un controlador BPID. Para esta optimización es necesario el ajuste y el estudio del algoritmo propuesto, así como de las técnicas de control, por lo tanto, los objetivos de este proyecto son:

- Estudiar y comprender el funcionamiento de las técnicas metaheurísticas para el desarrollo de algoritmos genéticos aplicados a la optimización.
- Comprender el concepto de las técnicas de control PID y sus derivaciones, en concreto del BPID, su diseño e implementación para problemas reales.
- Desarrollar el algoritmo genético RRA, evaluar su funcionamiento y adaptación a problemas de control. Aplicar dicho algoritmo a la optimización de los parámetros de controladores BPID y comprobar si su ajuste resulta beneficioso.

1.2 Entorno de aplicación – ambiente socioeconómico

Los primeros controladores PID empezaron con el diseño de los limitadores de velocidad aunque fue posteriormente cuando su uso se extendió al aplicarse para la dirección automática de barcos. El primer análisis teórico de un controlador PID fue publicado por el ingeniero ruso americano Nicolas Minorsky en 1922 [2].

Minorsky estaba diseñando sistemas de dirección automática para la Armada de los Estados Unidos, y basó sus análisis observando al timonel, notando así que el timonel controlaba la nave no solo por el error actual, sino también en los errores pasados así como en la tasa actual de cambio, logrando así que Minorsky desarrollara un modelo matemático para esto. Se realizaron pruebas del controlador en el USS New Mexico (BB-40), donde este se encargaba de controlar la velocidad angular del timón.

A partir del año 1955, se desarrollan los métodos temporales, con el objetivo de solucionar los problemas planteados en aplicaciones aerospaciales. Estos métodos reciben un fuerte impulso con el desarrollo de las computadoras digitales, que constituían la plataforma tecnológica necesaria para su implantación, prueba y desarrollo.

La programación evolutiva utilizó originalmente máquinas de estado finito para predecir los entornos y utilizó la variación y la selección para optimizar las lógicas predictivas. A principios de los años setenta, tras el trabajo de John Henry Holland, los algoritmos genéticos se hicieron importantes en la ingeniería computacional.

Éste se basó en los términos matemáticos con los que pueden describirse los mecanismos básicos de variación de las plantas y de los animales. A partir de ahí, ideó un sistema informático capaz de, gracias a pequeñas mejoras y variaciones, generar códigos nuevos.

Hoy en día, la evolución artificial está adquiriendo popularidad como una aplicación para construir diferentes tipos de hardware, desde motores, a circuitos así como también antenas.

Por ejemplo en radiocomunicaciones, las antenas evolucionadas son antenas diseñadas por un computador automático que usa los algoritmos evolucionarios imitando la evolución de Darwin.

El Evolvable Systems Group comenzó a estudiar algoritmos evolucionarios en el año 2001, construyendo las llamadas Yagi-Uda que se muestran en la Figura 1, las antenas que comúnmente se colocaban en el techo de las casas antes de la existencia de la televisión por cable.

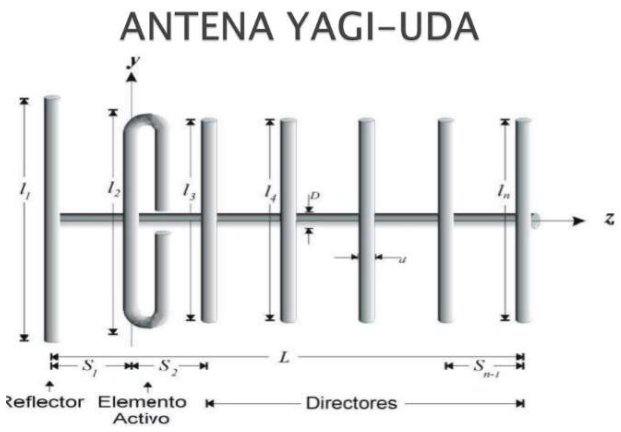


Figura 1. Representación de la forma de una antena Yagi-Uda [3]

El diseño más reciente es la antena utilizada por la NASA, la cual fue probada para ser lazada con un grupo de tres satélites en miniatura programados a orbitar la magnetosfera de la Tierra dentro de la misión 5 de Space Technology. Para fabricarla, se le presentaron a la máquina los requisitos necesarios para la composición de una antena, y el programa generó cientos de posibilidades. El resultado fue una antena aparentemente torcida, como se muestra en la Figura 2, pero que poseía la amplitud de banda que la NASA necesitaba [4].

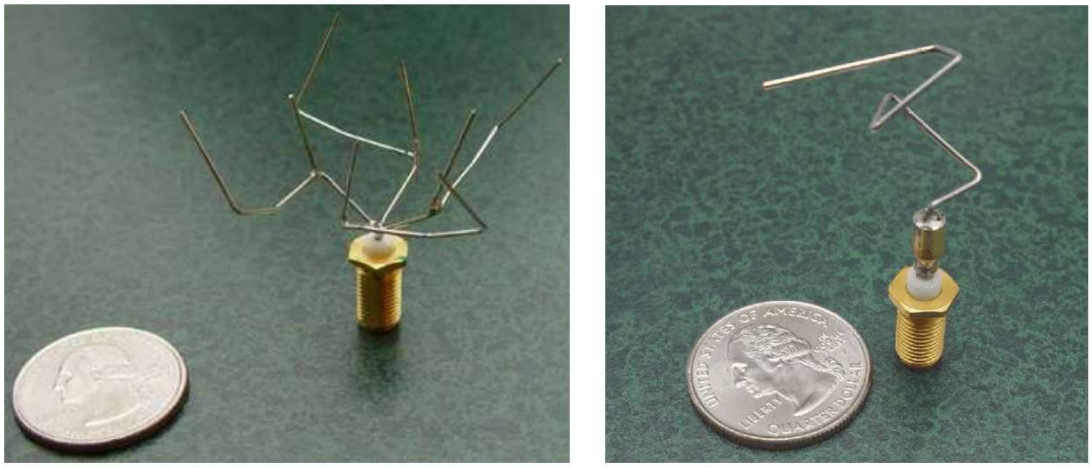


Figura 2. Fotografías del prototipo de antenas diseñadas por algoritmos evolutivos [4]

En cuanto al ámbito de la ingeniería de control, actualmente se utilizan estas técnicas en aplicaciones más cruciales tales como control de presión, flujo, fuerza, velocidad, y en muchas aplicaciones química. Además es utilizado en reguladores de velocidad de automóviles (control de crucero o cruise control).

El proyecto de investigación de vuelo Sistema inteligente de control de vuelo (IFCS), del Centro de Investigaciones de Vuelo Dryden de la NASA, se estableció para explotar un avance tecnológico revolucionario en los controles de vuelo de las aeronaves, que puede optimizar eficientemente el desempeño de las aeronaves como la de la Figura 3, tanto en condiciones normales como cuando ocurran desperfectos.

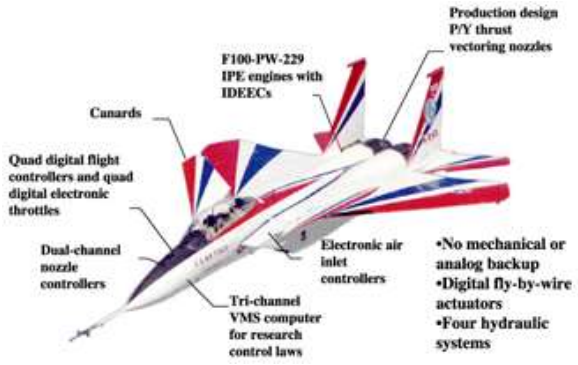


Figura 3 Maqueta de una de las aeronaves con el sistema de control de vuelo [5]

El equipo de IFCS ha integrado tecnologías de red neural con algoritmos de control de última generación, para identificar y responder correctamente a los cambios en la estabilidad y características de control de la aeronave, y poder hacer un ajuste inmediato para mantener el mejor desempeño de vuelo posible durante un desperfecto inesperado. El software de red neural adaptado "aprende" las nuevas características de vuelo, a bordo y en tiempo real, ayudando de este modo al piloto a mantener o recobrar el control y evitar un accidente potencialmente catastrófico [5].

1.3 Algoritmo de propagación de plantas (RRA)

Las plantas, al estar conectadas con la tierra a través de sus raíces, no se pueden mover para encontrar zonas con condiciones más favorables. A pesar de ello, algunas plantas (como las de la fresa) pueden propagarse a través de unos tallos como se muestra en la Figura 4. Estos tallos son progresivos, se producen en los pliegues de las hojas y crecen de la planta madre. En el segundo nudo del tallo se forman nuevas plantas, a partir de las cuales nuevamente vuelven a salir tallos generando así nuevas plantas hijas.

Figura 4. Fotografías de la propagación por tallos de la planta de la fresa [6]

Inicialmente, los tallos de las plantas echan pequeñas raíces en busca de agua y minerales y, tras ello, éstas desarrollan unos pelos radicales (Figura 5) hasta adquirir la suficiente madurez permitiendo así a las plantas hijas separarse de la

madre para continuar su vida individualmente como si de una planta madre se tratase.

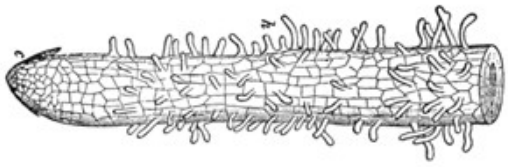


Figura 5. Representación esquemática de los pelos radicales en las raíces [7]

La reproducción de plantas como las de la fresa se puede pensar como un tipo de movimiento de las plantas puesto que tanto la madre como la hija tienen exactamente los mismos genes y son exactamente la misma planta, así como se ve en la Figura 6. Sin embargo, la planta madre termina muriendo antes que la planta hija siempre y cuando la hija no llegue a una zona con peores condiciones que las anteriores.

Figura 6. Fotografía de la reproducción de las plantas de la fresa [8]

Desde el punto de vista matemático, las plantas con tallos (como las de la fresa) ejecutan un tipo de optimización. De manera más precisa, este tipo de plantas ejecutan simultáneamente tanto una búsqueda global como una búsqueda local para buscar fuentes de recursos como agua y minerales al desarrollar los tallos y las raíces (con sus pelos radicales), respectivamente. Tanto los tallos como las raíces se crean aleatoriamente, pero cuando un tallo o una raíz llega a una zona con mejores condiciones, su correspondiente planta hija genera más tallos y raíces, lo que también afecta al crecimiento de la planta entera. Además, si una planta hija consigue llegar a un punto óptimo local, ésta genera más tallos y raíces que le ayude a alcanzar zonas más lejanas y escapar de ese punto (este proceso se denomina re-inicialización, el cual redistribuye los agentes en el dominio del problema).

Evidentemente, con el fin de llegar a un algoritmo de optimización numérica inspirado en plantas como las de la fresa se necesita modelar el comportamiento de esta planta con unas normas simples y explícitas. En este trabajo, se asume que el comportamiento de este tipo de plantas se puede modelar mediante los siguientes tres hechos:

- Cada planta madre se propaga a través de los tallos los cuales crecen aleatoriamente (búsqueda global de recursos).

- Cada planta madre desarrolla raíces y pelos radicales aleatoriamente (búsqueda local de recursos).

- Las plantas hijas que tienen acceso a zonas más ricas en recursos crecen más rápido y generan más tallos y raíces, y consecuentemente cubren zonas más amplias. Por otro lado, las plantas hijas que crecen en zonas más pobres tienen más probabilidad de morir.

Siguiendo estas premisas, en este algoritmo propuesto primeramente se genera una cantidad aleatoria de puntos (agentes computacionales) en el dominio del problema, los cuales pueden considerarse cada uno como una planta madre. Tras ello, en cada iteración cada planta madre genera un tallo y una raíz con sus pelos radicales (planta hija). Después, la función objetivo se evalúa en los nuevos puntos referidos a las plantas hijas, las cuales son seleccionadas mediante selección elitista y por rueda de ruleta para considerarse plantas madres de la siguiente iteración. Este proceso se repite hasta que una condición de terminación predeterminada se cumpla. La Figura 7 muestra el diagrama de flujo del algoritmo descrito.

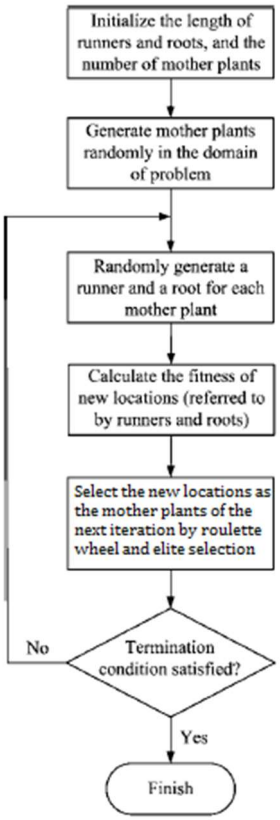


Figura 7. Diagrama de flujo del algoritmo RRA

1.4 Metodología

La metodología de este proyecto comienza por la investigación de documentación acerca de los algoritmos genéticos, con el objetivo de implementarlos posteriormente a técnicas de control bilineal.

Para este primer paso es necesaria la búsqueda de información relativa a los algoritmos metaheurísticos los cuales se clasifican en varios grupos, entre los que se encuentran los algoritmos genéticos además de otros tantos.

Una vez que se han adquirido los conocimientos acerca de los algoritmos genéticos y sus variaciones, es necesario comprender las diferentes técnicas de control y su funcionamiento.

En este paso se recauda la información necesaria relativa a los controladores PID y sus derivaciones, para llegar a introducir los sistemas de control bilineales BPID en los que se basa este proyecto.

Una vez comprendidos los conceptos de los algoritmos genéticos y los distintos tipos de controladores se procede al estudio del algoritmo RRA, el algoritmo que se quiere implementar en el proyecto para controladores BPID. Durante esta etapa se pretende comprender el funcionamiento del algoritmo y su programación para su posterior utilización a problemas reales de control.

Tras este estudio del algoritmo RRA, se hace especial hincapié en el estudio de los controladores BPID, repasando sus características y propiedades.

El siguiente paso a los fundamentos teóricos es la aplicación de los mismos a la práctica y analizar los resultados en tres problemas distintos:

1. Distancia mínima al punto.
2. Control de la posición de un motor DC mediante el BPID.
3. Control de la velocidad de un motor DC mediante el BPID.

El último paso consiste en sacar las conclusiones referidas al proyecto con respecto a los resultados obtenidos y comentar sus posibles aplicaciones futuras.

1.5 Estructura del documento

El documento se presenta estructurado en seis capítulos:

El capítulo 1 corresponde a la introducción del documento. Este capítulo se divide a su vez de varias secciones:

- Sección 1.1 Enfoque y objetivos del proyecto
- Sección 1.2 Entorno de aplicación – ambiente socio-económico
- Sección 1.3 Introducción del RRA
- Sección 1.4 Metodología
- Sección 1.5 Estructura del documento

El capítulo 2 contiene los fundamentos teóricos en los que se basa el proyecto, desde las técnicas de optimización metaheurísticas hasta las técnicas de control bilineal. Este capítulo queda dividido en siguientes secciones:

- Sección 2.1 Algoritmos metaheurísticos
    - Sección 2.1.1 Algoritmo genético
    - Sección 2.1.2 Evolución diferencial
    - Sección 2.1.3 Optimización biogeográfica (BBO)
    - Sección 2.1.4 Aproximación DE/BBO
- Sección 2.2 Técnicas de control
    - Sección 2.2.1 PID convencional
    - Sección 2.2.2 Control PIPD
    - Sección 2.2.3 Controladores BPID
    - Sección 2.2.4 Controladores PID basados en DE

En el capítulo 3 se explica el funcionamiento teórico y matemático del algoritmo propuesto. Este capítulo está dividido en dos secciones:

- Sección 3.1 Concepto general del algoritmo RRA
- Sección 3.2 Pseudo-código del RRA

En el capítulo 4 se realiza el estudio de la técnica de control bilineal BPID donde se repasan sus características y propiedades. Además se explica el proceso de linealización de sistemas no lineales a los que aplicar esta técnica.

- Sección 4.1 Concepto general de un controlador BPID

En el capítulo 5 se aplica el algoritmo RRA a problemas de control simulados en el software de programación Matlab y se evalúan los resultados experimentales. Este capítulo se divide en tres secciones:

- Sección 4.1 Control de la distancia al punto
- Sección 4.2 Control de la posición de un motor de corriente continua
- Sección 4.3 Control de la velocidad de un motor de corriente continua

En el capítulo 6 se recogen las conclusiones y el análisis de los resultados obtenidos experimentalmente en el apartado anterior.

Por último se incluye en un Anexo el presupuesto del proyecto con los gastos desglosados y el código de programación de las distintas funciones empleadas en este proyecto.

Capítulo 2. Fundamentos teóricos

Este capítulo recoge la información y los conocimientos necesarios tanto de los algoritmos de optimización, profundizando desde los algoritmos metaheurísticos hasta los algoritmos genéticos y la evolución diferencial; como de las técnicas de control.

En la Sección 2.1 se investiga sobre los algoritmos metaheurísticos, su funcionamiento, su origen, sus características, técnicas de empleo y clasificación de las mismas.

Por otro lado, el apartado 2.2 trata sobre las distintas técnicas de control, desde el PID convencional hasta la introducción de los reguladores bilineales.

2.1 Algoritmos metaheurísticos

En las ciencias de computación, dos objetivos fundamentales son encontrar algoritmos con buenos tiempos de ejecución y buenas soluciones, usualmente las óptimas. Una heurística es un algoritmo que abandona uno o ambos objetivos; por ejemplo, normalmente encuentran buenas soluciones, aunque no hay pruebas de que la solución no pueda ser arbitrariamente errónea en algunos casos; o se ejecuta razonablemente rápido, aunque no existe tampoco prueba de que siempre será así. Las heurísticas generalmente son usadas cuando no existe una solución óptima bajo las restricciones dadas (tiempo, espacio, etc.), o cuando no existe del todo [9].

Un método heurístico para resolver problemas generales de tipo computacional es la metaheurística, usando los parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera que se espera eficiente [10].

Las metaheurísticas generalmente se aplican a problemas que no tienen un algoritmo o heurística específica que dé una solución satisfactoria; o bien cuando no es posible implementar ese método óptimo. La mayoría de las metaheurísticas tienen como objetivo los problemas de optimización combinatoria. El objetivo de la optimización combinatoria es encontrar un objeto matemático finito (por ejemplo, un vector de bits) que maximice (o minimice, dependiendo del problema) una función especificada por el usuario. A estos objetos se les suele llamar estados, y al conjunto de todos los estados candidatos se le llama espacio de búsqueda. La naturaleza de los estados y del espacio de búsqueda son normalmente específicos del problema.

La función a optimizar se le llama función objetivo, y se da al usuario como un procedimiento que evalúa la aptitud del estado actual o la función. Dependiendo de la metaheurística, el usuario puede tener que dar otras funciones de aptitud que produzcan un nuevo estado, generan variantes del estado actual, elijan un estado entre varios, aporten valores máximos o mínimos para la función

objetivo en un conjunto de estados

Además, una metaheurística puede guardar información del óptimo actual, escogiendo el estado óptimo entre todos los óptimos actuales obtenidos en varias etapas del algoritmo.

Normalmente, las metaheurísticas están diseñadas de manera que puedan ser interrumpidas por un tiempo máximo especificado por el usuario, dado que el número de candidatos puede ser muy grande. Si no se interrumpen, algunas metaheurísticas exactas examinaran todos los candidatos, y usarán métodos heurísticos sólo para escoger el orden de la enumeración; de hecho, siempre devolverán un óptimo real, si el tiempo máximo es lo suficientemente grande. En cambio, otras metaheurísticas dan sólo una garantía probabilística pobre de poder alcanzar el óptimo, de manera que cuando el tiempo máximo se aproxima a infinito, la probabilidad de examinar cada candidato tiende a 1.

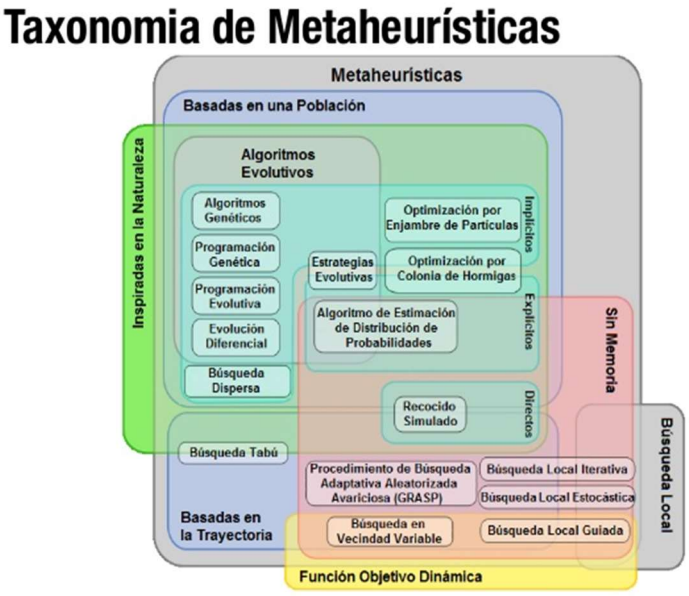


Figura 8. Cuadro clasificativo de los metaheurísticos [10]

Hay un número enorme de variables e híbridos propuestos, como se muestra en la Figura 8, y muchas más metaheurísticas han sido probadas en problemas específicos. Este es un campo en investigación, con un gran número de publicaciones en revistas, un gran número de investigadores y usuarios, además de un gran número de aplicaciones.

Técnicas metaheurísticas

La aplicación de las técnicas metaheurísticas es especialmente interesante en caso de problemas de optimización combinatoria: problemas en las que las variables de decisión son enteras (o discretas, al menos) en las que, generalmente, el espacio de soluciones está formado por disposiciones de valores de dichas variables. Sin embargo, las técnicas metaheurísticas se pueden aplicar también a

problemas de otro tipo, como con variables continuas, por ejemplo.

La lógica de las técnicas metaheurísticas comienza con una solución (o conjunto de soluciones) que típicamente no es óptima. A partir de ella se obtienen otras parecidas, de entre las cuales se elige una que satisface algún criterio, a partir de la cual comienza de nuevo el proceso. Este proceso se detiene cuando se cumple alguna condición establecida previamente.

Las técnicas metaheurísticas más extendidas son: los algoritmos genéticos, la búsqueda tabú, el recocido simulado, la búsqueda "scatter", las colonias de hormigas, etc; y todas ellas tienen las siguientes características [11]:

- Son ciegas, no saben si llegan a la solución óptima. Por lo tanto, se les debe indicar cuándo deben detenerse.
- Son algoritmos aproximativos y, por lo tanto, no garantizan la obtención de la solución óptima.
- Aceptan ocasionalmente malos movimientos (es decir, se trata de procesos de búsqueda en los que cada nueva solución no es necesariamente mejor –en términos de la función objetivo– que la inmediatamente anterior). Algunas veces aceptan, incluso, soluciones no factibles como paso intermedio para acceder a nuevas regiones no exploradas.
- Son relativamente sencillos. Todo lo que se necesita es una representación adecuada del espacio de soluciones, una solución inicial (o un conjunto de ellas) y un mecanismo para explorar el campo de soluciones.
- Son generales. Prácticamente se pueden aplicar en la resolución de cualquier problema de optimización de carácter combinatorio. Sin embargo, la definición de la técnica será más o menos eficiente en la medida en que las operaciones tengan relación con el problema considerado.
- La regla de selección depende del instante del proceso y de la historia hasta ese momento. Si en dos iteraciones determinadas, la solución es la misma, la nueva solución de la siguiente iteración no tiene por qué ser necesariamente la misma. En general, no lo será.

A pesar de que estas técnicas son relativamente recientes, sus campos de aplicación son numerosos (electrónica, telecomunicaciones, electromagnetismo, etc.), y entre ellos se encuentra el de la Ingeniería de Control.

2.1.1 Algoritmos genéticos

Un algoritmo genético (AG) es un método de búsqueda que imita la teoría de la evolución biológica de Darwin para la resolución de problemas. Para ello, se parte de una población inicial de la cual se seleccionan los individuos más capacitados para luego reproducirlos y mutarlos para finalmente obtener la siguiente generación de individuos que estarán más adaptados que la anterior generación.

Así mismo, se puede decir que los AG son algoritmos de búsqueda basados en los mecanismos de selección natural y genética natural. Combinan la supervivencia de los más compatibles entre las estructuras de cadenas, con una estructura de información ya aleatorizada, intercambiada para construir un algoritmo de búsqueda con algunas de las capacidades de innovación de la búsqueda humana [12].

Básicamente, el AG funciona de la siguiente manera: en cada generación, se crea un conjunto nuevo de "criaturas artificiales" (cadenas) utilizando bits y partes más adecuadas del progenitor. Esto involucra un proceso aleatorio que no es, en absoluto, simple. La novedad que introducen los AG es que explotan eficientemente la información histórica para especular sobre nuevos puntos de búsqueda, esperando un funcionamiento mejorado.

La evolución, tal y como la conocemos, es básicamente un método de búsqueda entre un número enorme de posibles "soluciones". En biología las posibilidades están formadas por un conjunto de secuencias genéticas posibles, y las soluciones deseadas, por organismos capaces de sobrevivir y reproducirse en sus entornos. La evolución puede verse, asimismo, como un modo de "diseñar" soluciones a problemas complejos, con la capacidad de innovar. Estos son los motivos por los que los mecanismos evolutivos son una fuente de inspiración para los algoritmos de búsqueda.

El buen funcionamiento de un organismo biológico depende de muchos criterios, que además varían a medida que el organismo evoluciona, de modo que la evolución está "buscando" continuamente entre un conjunto cambiante de posibilidades. Por ello, se puede considerar como un método de búsqueda paralelo, ya que evalúa y cambia muchas especies al mismo tiempo [13].

Para terminar, las reglas de la evolución, aunque de alto nivel, son simples: las especies evolucionan mediante variaciones aleatorias (vía mutaciones, recombinaciones, etc.) seguidas por la selección natural, donde el mejor tiende a sobrevivir y reproducirse, propagando así su material genético a posteriores generaciones, como se aprecia en la Figura 9.

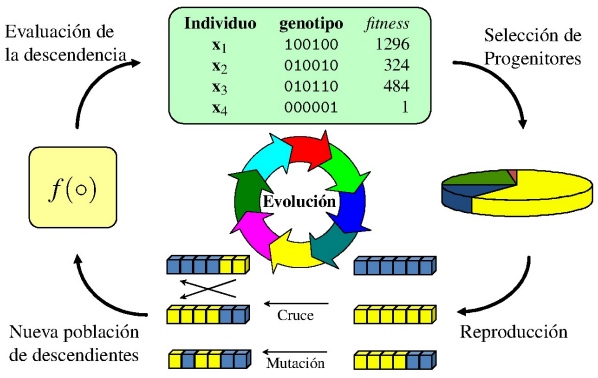


Figura 9. Representación gráfica del algoritmo genético [14]

Como se ha dicho, los AG están basados en la observación de la evolución natural de las especies. Existen, por lo tanto, analogías entre la nomenclatura propia de la Biología y la que se emplea en la técnica metaheurística de los AG.

Los cromosomas de los seres vivos contienen la información de los mismos que se llama genotipo. La descodificación (realizada por diferentes enzimas) del genotipo de cada individuo da lugar a un conjunto de características (llamado fenotipo), lo que le confiere al individuo unas determinadas condiciones de adaptación en función del entorno en el que se encuentre.

En los algoritmos genéticos, el equivalente del genotipo es una cadena de caracteres. El equivalente del fenotipo es la solución que resulta de la descodificación de la cadena anterior y, finalmente, la función objetivo hace las veces de entorno y permite evaluar la aptitud de las soluciones. El significado del término 'individuo' es distinto en el contexto de los AG y en el ámbito biológico. En términos biológicos, un individuo es un animal o vegetal de determinada especie. Sin embargo, al hablar de un individuo en los AG a una cadena de caracteres que representan una solución o un conjunto de ellas. El conjunto de individuos constituye una población. La figura 10 ilustra la relación entre los AG y la biología [15]:

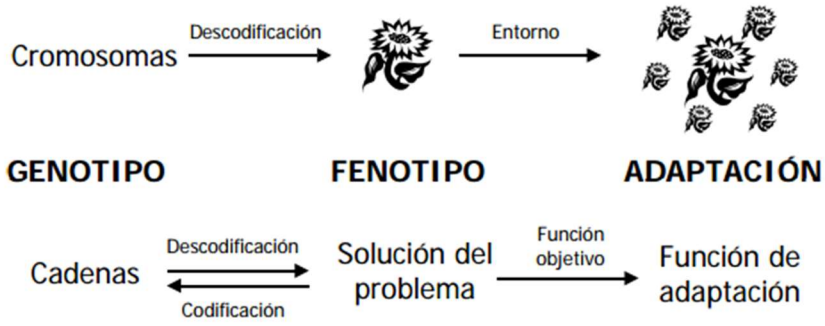


Figura 10. Analogía entre los algoritmos genéticos y la biología [15]

La forma más simple de algoritmo genético utiliza tres tipos de operadores: selección, cruce y mutación.

Selección

Es necesario hacer una selección con los individuos más capacitados para que éstos sean los que se reproduzcan con más probabilidad de acuerdo con la teoría de Darwin en la cual los más capacitados son los que deben sobrevivir y crear una nueva descendencia mejor adaptada. Por lo tanto una vez evaluado cada cromosoma y obtenida su puntuación, se tiene que crear la nueva población teniendo en cuenta que los buenos rasgos de los mejores se transmitan a ésta. Esta selección se puede realizar de varias formas como se verá a continuación [16]:

- Selección por Rueda de Ruleta: Se crea para esta selección una ruleta con los cromosomas presentes en una generación. Cada cromosoma tendrá una parte de esa ruleta mayor o menor en función a la puntuación que tenga cada uno. Se hace girar la ruleta y se selecciona el cromosoma en el que se para la ruleta. Obviamente el cromosoma con mayor puntuación saldrá con mayor probabilidad. En caso de que las probabilidades difieran mucho, este método de selección dará problemas puesto que si un cromosoma tiene un 90% de posibilidades de ser seleccionado, el resto apenas saldrá lo que reduciría la diversidad genética.
- Selección por Rango: En este método a cada cromosoma se le asigna un rango numérico basado en su aptitud y la selección se realiza en base a este ránking. La diferencia entre el caso anterior y este se puede observar en el ejemplo gráfico de la Figura 11.

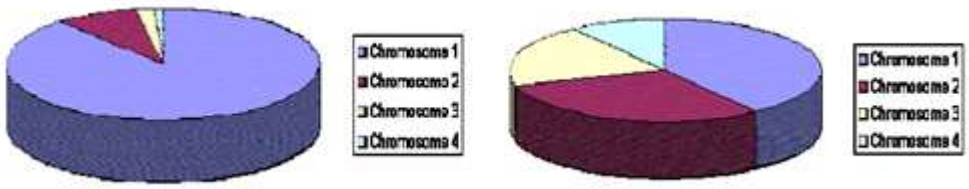


Figura 11. Selección por Rueda de Ruleta (Izq) y por Rango (Der) [16]

- Selección Elitista: En ciertas ocasiones puede suceder que tras el cruce y la mutación, perdamos el cromosoma con mejor adaptación. Este método de selección copia el mejor cromosoma o alguno de los mejores en la nueva población. El resto se realiza de la misma forma que hemos visto anteriormente. El elitismo puede mejorar el funcionamiento de los algoritmos genéticos al evitar que se pierda la mejor solución.
- Selección por Torneo: Se escogen de forma aleatoria un número de individuos de la población, y el que tiene puntuación mayor se reproduce, sustituyendo su descendencia al que tiene menor puntuación.

Cruce

Este operador es el más importante de esta técnica metaheurística. El objetivo del cruce es combinar elementos de información de diferentes individuos, de modo que las características interesantes que estaban dispersas en diferentes individuos queden reunidas en uno nuevo, confiando en que los individuos obtenidos de esta manera representen soluciones de mejor calidad.

La forma en que se realiza el cruce depende del tipo de representación que se escoja. Para ilustrar algunas posibilidades se supone que la representación de las soluciones de un determinado problema es de tipo binario y es una cadena de ocho elementos. Los individuos A y B han sido seleccionados para someterlos al operador del cruce [15]:

Individuo A: 1 1 1 0 1 0 0 1

Individuo B: 0 1 0 0 1 1 0 0

Se pueden describir los siguientes tipos de cruce (que no son los únicos):

- Operador de cruce simple: Según este tipo se cruce, se selecciona un punto de la cadena en cada uno de los padres y se generan nuevos individuos combinando las partes que generan los cortes anteriores, como aparece en la Figura 12.

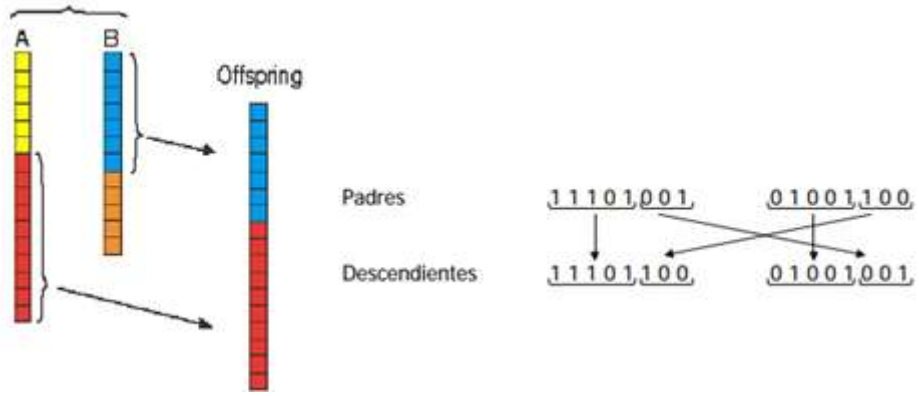


Figura 12. Representación del cruce simple [15,16]

- Operador con dos puntos de corte: Es análogo al anterior, salvo que se seleccionan dos puntos de corte y los padres intercambian los elementos de la cadena que quedan entre dichos puntos para generar los descendientes (Figura 13).

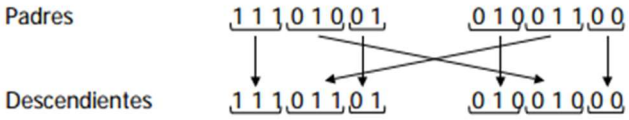


Figura 13. Representación del cruce con dos puntos [15]

- Operador de cruce conforme a una máscara de cruce: En este caso, los genes de los descendientes se obtienen de acuerdo con el criterio dado por una máscara. La máscara, en el ejemplo de la Figura 14, es una cadena de ceros y unos. Para cada posición de la descendencia se tomará el gen del padre 1 si el valor de la máscara para dicha posición es 1 y del padre 2 si el valor del gen es 0.

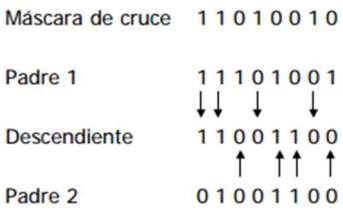


Figura 14. Representación del cruce con máscara [15]

- Cruce Aritmético: Los progenitores se recombinan según algún operador aritmético para generar su descendiente. En la figura 15 se muestran dos ejemplos:

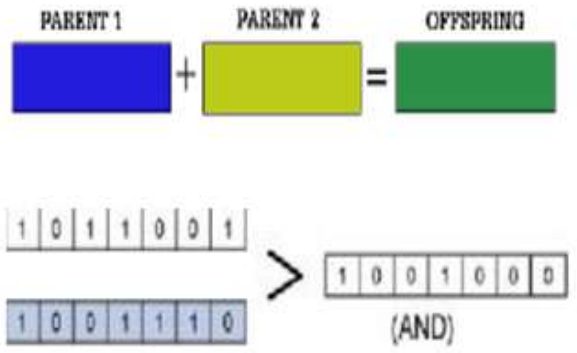


Figura 15. Representación del cruce aritmético: Arriba mediante combinación y abajo mediante el operando AND [16]

Mutación

Con los operadores anteriores se obtienen individuos que combinan rasgos que están presentes en los individuos de la población. Sin embargo, pueden existir características que quedan sin explorar si sólo se emplean los operadores anteriores. Con la mutación se introducen modificaciones en los individuos de la descendencia.

En el ejemplo de la Figura 16, la mutación consistiría en transformar un cero en uno (o viceversa) con una probabilidad determinada:

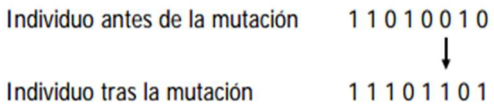


Figura 16. Representación de un ejemplo de mutación [15]

Si la probabilidad con la que tiene lugar la mutación en muy pequeña, es difícil que vuelvan a aparecer características que han sido abandonadas por el operador de cruce.

Si la probabilidad de mutación es elevada, es probable que alguna de las características adecuadas de la solución se pierda y la técnica se desvíe de regiones potencialmente interesantes sin haber explorado con suficiente profundidad las regiones visitadas.

En la figura 17 se pueden ver otros ejemplos de mutación en diferentes genes de una misma configuración.

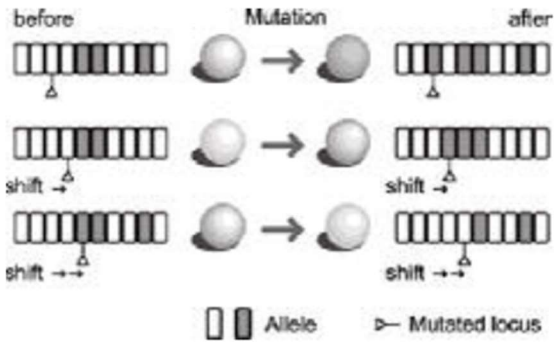


Figura 17. Representación de tres mutaciones distintas sobre una misma configuración [16]

En el flujograma de la Figura 18 se resume el funcionamiento de los AG [17].

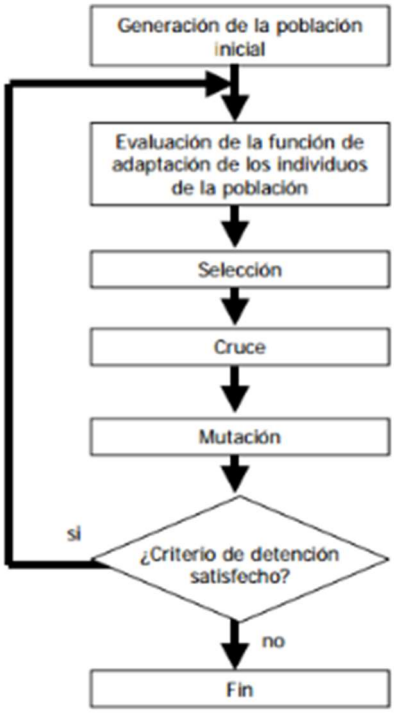


Figura 18. Diagrama de flujo general del algoritmo genético [17]

Aplicaciones del algoritmo genético

- Optimización: Se trata de un campo especialmente abonado para el uso de los AG, puesto que fueron la fuente de inspiración para los creadores estos algoritmos. Los AG se han utilizado en numerosas tareas de optimización, incluyendo la optimización numérica, y los problemas de optimización combinatoria.
- Programación automática: Los AG se han empleado para desarrollar programas para tareas específicas, y para diseñar otras estructuras computacionales tales como el autómata celular, y las redes de clasificación.
- Aprendizaje máquina: Los AG se han utilizado también en muchas de estas aplicaciones, tales como la predicción del tiempo o la estructura de una

proteína. Han servido asimismo para desarrollar determinados aspectos de sistemas particulares de aprendizaje, como pueda ser el de los pesos en una red neuronal, las reglas para sistemas de clasificación de aprendizaje o sistemas de producción simbólica, y los sensores para robots.

- Economía: En este caso, se ha hecho uso de estos AG para modelizar procesos de innovación, el desarrollo estrategias de puja, y la aparición de mercados económicos.
- Sistemas inmunes: A la hora de modelizar varios aspectos de los sistemas inmunes naturales, incluyendo la mutación somática durante la vida de un individuo y el descubrimiento de familias de genes múltiples en tiempo evolutivo, ha resultado útil el empleo de esta técnica.
- Ecología: En la modelización de fenómenos ecológicos tales como las carreras de armamento biológico, la coevolución de parásito-huesped, la simbiosis, y el flujo de recursos.
- Sistemas sociales: En el estudio de aspectos evolutivos de los sistemas sociales, tales como la evolución del comportamiento social en colonias de insectos, y la evolución de la cooperación y la comunicación en sistemas multi-agentes.

Aunque esta lista no es, en modo alguno, exhaustiva, sí transmite la idea de la variedad de aplicaciones que tienen los AG. Gracias al éxito en estas y otras áreas, los AG han llegado a ser un campo puntero en la investigación actual.

Uno de los campos relacionados con los AG son los algoritmos evolutivos, una estrategia de computación evolutiva que engloba a los AG así como la evolución diferencial. La evolución diferencial es otra de las técnicas de optimización que se estudian en este trabajo.

2.1.2 Evolución diferencial

La Evolución Diferencial (DE) es un método de optimización perteneciente a la categoría de algoritmos evolutivos, aplicado en la resolución de problemas complejos. Al igual que otros algoritmos de esta categoría, la DE mantiene una población de soluciones candidatas, las cuales se recombinan y mutan para producir nuevos individuos los cuales serán elegidos de acuerdo al valor de su función de desempeño. Lo que caracteriza a la DE es el uso de vectores de prueba, los cuales compiten con los individuos de la población actual a fin de sobrevivir [18].

La población mejora iterativamente, con los tres operadores básicos: mutación, cruce y selección.

Inicialización

La población se inicializa aleatoriamente generando individuos dentro del dominio del problema

$$X_{ij}^{0} = X_j^{min} + rand * \left(X_j^{max} - X_j^{min}\right)$$
$$i = 1, 2, 3, \dots, Np;\ \ j = 1, 2, 3, \dots, D$$

donde la función "$rand$" genera valores aleatorios uniformemente en el intervalo [0, 1] ; Np es el tamaño de la población; D es el número de variables. X^{min} y X^{max} son límites inferior y superior respectivamente de las variables.

Mutación

Como un paso para generar descendencia, se aplican las operaciones de mutación. La mutación adquiere un rol bastante importante en el ciclo reproductor. Las operaciones de mutación crean vectores mutados X_i^k al perturbar unos vectores aleatoriamente seleccionados X_a^k con la diferencia entre otros dos vectores aleatoriamente seleccionados X_b^k y X_c^k en la kº iteración, como sigue en la siguiente ecuación:

$$X_i^k = X_a^k \pm F * \left(X_b^k - X_c^k\right);\quad i = 1, 2, \dots, Np$$

donde F ∈ [0, 2] es conocido como un "Factor escalar" usado para controlar la cantidad de perturbaciones en el proceso de mutación.

Cruce

El cruce representa un caso típico de intercambio de genes. El primer ensayo hereda genes con cierta probabilidad. El vector padre se mezcla con el vector mutado para crear un vector de ensayo, siguiendo la siguiente ecuación:

$$X_{ij}^{//k} = \begin{cases} X_{ij}^{/k}, & if\ randj < Cr\ or\ j = q \\ X_{ij}^{k}, & otherwise \end{cases}$$

donde X_{ij}^k $X_{ij}^{/k}$ $X_{ij}^{//k}$son el jº individuo del iº vector objetivo, el vector mutado, y el vector de ensayo en la kº iteración, respectivamente. q es un índice elegido aleatoriamente ϵ (j = 1, 2, …, D) del vector mutante incluso si $Cr = 0$. Cr ϵ [0, 1] es la constante de cruce que controla la diversidad de la población y ayuda al algoritmo a escapar de un óptimo local.

Selección

El proceso de selección se usa entre un conjunto de vectores de ensayo y el vector objetivo actualizado para elegir al mejor candidato. La selección se realiza comparando los valores de la función objetivo tanto del vector objetivo como del vector de ensayo. La operación de selección se ejecuta de la siguiente manera:

$$X_i^{k+1} = \begin{cases} X_i^{//k}, & if\ f(X_i^{//k}) \le f(X_i^k) \\ X_i^k, & otherwise \end{cases};\quad i = 1, 2, \dots, Np$$

2.1.3 Optimización biogeográfica (BBO)

La Biogeografía describe cómo las especies migran de una isla a otra, cómo surge una nueva especie, y cómo se extinguen éstas. Un hábitat es cualquier isla (área) que está geográficamente aislada de otras islas. Hábitats con un alto HSI tienden a tener mayor número de especies, mientras que aquellas con bajo HSI tienen menos especies. Hábitats con un alto HSI tienen un ritmo menor de inmigración de especies porque ya están saturadas de especies. Por la misma razón, hábitats con alto HSI tienen un mayor ritmo de emigración. Hábitats con un bajo HSI tienen un alto ritmo de inmigración debido a sus poblaciones dispersas.

El ritmo de emigración funciona de manera similar. La emigración en la BBO no significa que la isla emigratoria pierda características. La peor solución se asume que tiene las peores características; en consecuencia, tiene un ritmo de emigración muy bajo y una pequeña oportunidad de compartir sus características. La solución con mejores características también tiene la mayor probabilidad de compartirlas. Este enfoque es conocido como la optimización biogeográfica (BBO) [18].

Matemáticamente el concepto de inmigración y emigración puede ser representado como una moda probabilística. Considerando P_s como la probabilidad de que el hábitat contenga exactamente S especies en t, P_s cambia con el tiempo de la siguiente manera:

$$P_s(t+\Delta t) = P_s(t)(1-\lambda_s\Delta t-\mu_s\Delta t) + P_{s-1}\lambda_{s-1}\Delta t + P_{s+1}\mu_{s+1}\Delta t$$

donde λ_s y μ_s son los ritmos de inmigración y emigración cuando hay S especies en el hábitat. Esta ecuación es válida ya que para tener S especies en el tiempo $(t + \Delta t)$, una de las siguientes condiciones se de cumplir:

1) Que haya S especies en el tiempo t, y no ocurran ni inmigraciones ni emigraciones entre t y $t + \Delta t$;
2) Que hayan $S - 1$ especies en el tiempo t y una especie inmigre;
3) Que hayan $S + 1$ especies en el tiempo t, y una especie emigre.

Si el tiempo Δt es lo suficientemente pequeño como para que la probabilidad de que más de una inmigración o emigración sea ignorada entonces obtenemos la siguiente ecuación:

$$\dot{P}_s \begin{cases} -(\lambda_s+\mu_s)P_s + \mu_{s+1}P_{s+1} & S = 0 \\ -(\lambda_s+\mu_s)P_s + \lambda_{s-1}P_{s-1} & \\ +\mu_{s+1}P_{s+1} & 1 \le S \le S_{max}-1 \\ -(\lambda_s+\mu_s)P_s + \lambda_{s-1}P_{s-1} & S = S_{max} \end{cases}$$

Las ecuaciones de los ritmos de inmigración y emigración para el número k de especies se pueden escribir de esta manera:

$$\mu_k = \frac{Ek}{n} \qquad\qquad \lambda_k = I(1-\frac{k}{n})$$

2.1.4 Aproximación DE / BBO

La evolución diferencial ha conseguido obtener mejores soluciones y más rápidas, cumpliendo las restricciones, tanto para sistemas unimodales o multimodales, usando distintas estrategias de cruce. Pero cuando la complejidad del sistema y su tamaño incrementa, el método de evolución diferencial no es capaz de trazar todas las variables desconocidas juntas.

Debido a la presencia de una operación de cruce en los algoritmos evolutivos, muchas soluciones cuya aptitud son inicialmente buenas, a veces pierden su calidad en siguientes estados del proceso.

En BBO no hay algo como una operación de cruce; las soluciones se ajustan finamente gradualmente mientras el proceso continúa por la operación de migración. Esto da una ventaja al BBO sobre otras técnicas evolutivas. En una palabra, DE tiene una buena habilidad de exploración en encontrar una región de mínimo global. Así mismo, BBO tiene una gran habilidad de explotación en un problema de optimización global.

Para utilizar ambas propiedades del DE y BBO para soluciones de problemas de optimización complejos, se ha desarrollado una técnica híbrida llamada DE/BBO [19].

Operador de la migración híbrida

El operador de la migración híbrida es el paso más importante en el algoritmo DE/BBO. En este algoritmo la descendencia Ui toma nuevas características de diferentes operandos. Éstos son la operación de mutación de la DE, la operación de migración de la BBO y los padres Xi correspondientes de la descendencia.

La idea principal del propuesto operador de migración híbrida se basa en dos consideraciones. Debido a la hibridación se podrían destruir menos soluciones buenas, mientras que las soluciones más pobres pueden aceptar muchas nuevas características provenientes de las buenas soluciones.

Funcionamiento del DE/BBO

Al introducir el previo operador de migración híbrido del BBO en el DE se ha desarrollado un nuevo algoritmo conocido como DE/BBO. Su estructura es muy simple y se detalla de la siguiente manera:

1) *Inicialización de la población P*
2) *Evaluar la aptitud de cada individuo en P*
3) *Mientras que el criterio de terminación no se cumpla*
    a) *Para cada individuo calcular la probabilidad de contar especies*
    b) *Para cada individuo calcular los ritmos de inmigración y emigración*

c) *Modificar la población con la migración híbrida*
d) *Para cada individuo*
   i) *Evaluar su descendencia*
   ii) *Si ésta es mejor, entonces Pi = Ui*
   iii) *end*
e) *end*

4) *end*

2.2 Técnicas de control

2.2.1 PID convencional

Esta estrategia de control es la más tradicional. Se trata de un mecanismo de control por realimentación usado en sistemas de control industrial. El algoritmo de control consiste en tres parámetros: el proporcional, el integral, y el derivativo. El valor Proporcional depende del error actual. El Integral depende de los errores pasados y el Derivativo es una predicción de los errores futuros. La suma de estas tres acciones es usada para ajustar al proceso por medio de un elemento de control como la posición de una válvula de control o la potencia suministrada a un calentador.

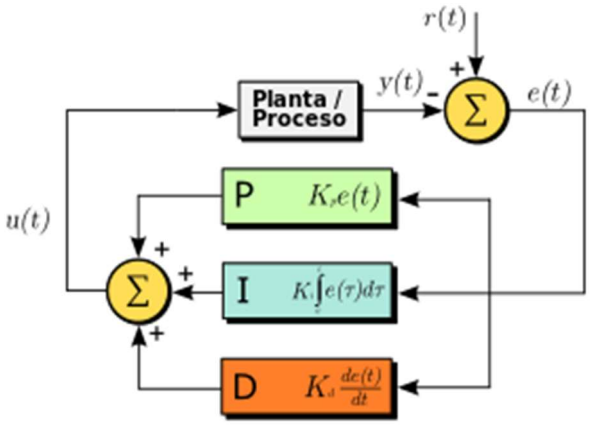


Figura 19. Representación esquemática de un controlador PID [20]

Ajustando estas tres variables en el algoritmo de control del PID, como se representa en la Figura 19, el controlador puede proveer una acción de control diseñado para los requerimientos del proceso en específico. La respuesta del controlador puede describirse en términos de la respuesta del control ante un error, el grado el cual el controlador sobrepasa el punto de ajuste, y el grado de oscilación del sistema.

La acción del controlador viene determinada por la siguiente ecuación, que es la suma de los tres tipos de acción del PID:

$$K = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

Algunas aplicaciones pueden sólo requerir de uno o dos modos de los que provee este sistema de control. Un controlador PID puede ser llamado también PI, PD, P o I en la ausencia de las acciones de control respectivas. Los controladores PI son particularmente comunes, ya que la acción derivativa es muy sensible al ruido, y la ausencia del proceso integral puede evitar que se alcance al valor deseado debido a la acción de control [20].

Algunas de las aplicaciones más comunes son:

- Lazos de temperatura (aire acondicionado, calentadores, refrigeradores, etc.)
- Lazos de nivel (nivel en tanques de líquidos como agua, lácteos, mezclas, crudo, etc.)
- Lazos de presión (para mantener una presión predeterminada en tanques, tubos, recipientes, etc.)

Para el correcto funcionamiento de un controlador PID que regule un proceso o sistema se necesita, al menos:

1. Un sensor, que determine el estado del sistema (termómetro, caudalímetro, manómetro, etc).
2. Un controlador, que genere la señal que gobierna al actuador.
3. Un actuador, que modifique al sistema de manera controlada (resistencia eléctrica, motor, válvula, bomba, etc).

El sensor proporciona una señal analógica o digital al controlador, la cual representa el punto actual en el que se encuentra el proceso o sistema. La señal puede representar ese valor en tensión eléctrica, intensidad de corriente eléctrica o frecuencia. En este último caso la señal es de corriente alterna, a diferencia de los dos anteriores, que también pueden ser con corriente continua.

El controlador recibe una señal externa que representa el valor que se desea alcanzar. Esta señal recibe el nombre de punto de consigna (o punto de referencia, valor deseado o *set point)*, la cual es de la misma naturaleza y tiene el mismo rango de valores que la señal que proporciona el sensor. Para hacer posible esta compatibilidad y que, a su vez, la señal pueda ser entendida por un humano, habrá que establecer algún tipo de interfaz (HMI - Human Machine Interface), son pantallas de gran valor visual y fácil manejo que se usan para hacer más intuitivo el control de un proceso.

El controlador resta la señal de punto actual a la señal de punto de consigna, obteniendo así la señal de error, que determina en cada instante la diferencia que hay entre el valor deseado (consigna) y el valor medido. La señal de error es

utilizada por cada uno de los 3 componentes del controlador PID. Las 3 señales sumadas, componen la señal de salida que el controlador va a utilizar para gobernar al actuador. La señal resultante de la suma de estas tres se llama variable manipulada y no se aplica directamente sobre el actuador, sino que debe ser transformada para ser compatible con el actuador utilizado.

Mientras que los controladores PID son aplicables a la mayoría de los problemas de control, puede ser pobres en otras aplicaciones.

Los controladores PID, cuando se usan solos, pueden dar un desempeño pobre cuando la ganancia del lazo del PID debe ser reducida para que no se dispare u oscile sobre el valor del *"setpoint"*. El desempeño del sistema de control puede ser mejorado combinando el lazo cerrado de un control PID con un lazo abierto.

Otro problema que posee el PID es que es lineal. Principalmente el desempeño de los controladores PID en sistemas no lineales es variable.

2.2.2 Control PIPD

Este modelo de control usa una estrategia de conmutación que cambia entre dos controladores: un controlador PD y un controlador PI con retroalimentación (ff+PI) [21]. La selección del controlador a usar en cada momento se hace mediante una señal de referencia. El controlador cambia automáticamente entre el controlador PD y el ff+PI dependiendo de su entrada. Si el actuador tiene que mantener una posición fija según una referencia de posición de entrada, entonces actúa el controlador PD. Cuando se da una nueva posición de referencia y el actuador tiene que contraerse o expandirse (por ejemplo) para llegar a la nueva posición, entonces el control cambia al controlador ff+PI.

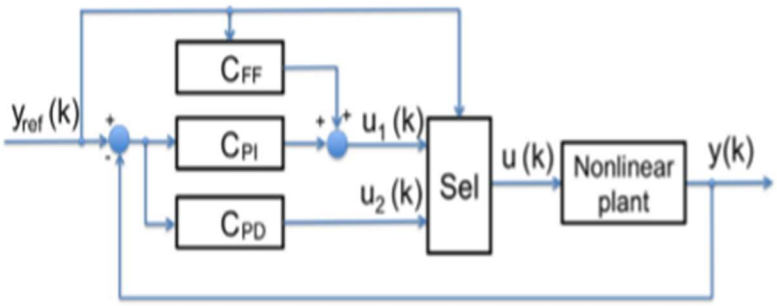


Figura 20. Representación esquemática de un controlador PIPD [22]

La arquitectura de control se muestra en la figura 20. El controlador PD es C_{PD} y el regulador PI es C_{PI}. C_{FF} representa la compensación introducida por el componente retroalimentado. Este enfoque se rige por las siguientes fórmulas y condiciones.

$$C_{FF} = \begin{cases} 0; & if\ y_{ref}(k-1) \\ i_{FF}; & otherwis \end{cases}$$

Donde i_{FF} es, por ejemplo, la corriente que se suma cuando sea necesario.

$$C_{PD} = K_{P_{PD}} e(k) + K_{d_{PD}} \frac{e(k) - e(k-1)}{T_s}$$

$$C_{PI} = K_{P_{PI}} e(k) + K_{i_{PI}} \int_0^k e(\tau) d\tau$$

El bloque selector conmuta la salida del controlador según la siguiente condición:

$$u(k) = \begin{cases} u_1(k); & if\ y_{ref}(k) \neq y_{ref}(k-1) \\ u_2(k); & if\ y_{ref}(k) = y_{ref}(k-1) \end{cases}$$

2.2.3 Controladores BPID

Dentro de las clases de sistemas no lineales, los sistemas bilineales representan una subclase que se define como lineal tanto en el estado como en el control, y la no linealidad (bilinearidad) ocurre como un producto entre el estado y el control [23]. La representación en el espacio de estados de un sistema bilineal de una sola entrada y una sola salida (SISO) continua es dada por:

$$\dot{x}(t) = Ax(t) + bu(t) + u(t)Nx(t)$$

Este tipo de sistemas tienen propiedades interesantes. El tipo de sistema de la ecuación anterior es un sistema simple y parecido a un sistema lineal. Este hecho permite la aplicación de diversas técnicas y procedimientos de análisis que se usan para sistemas lineales. Además, la estructura no lineal ofrece otras características importantes [22].

El modelo de control bilineal se puede utilizar para el estudio de las leyes básicas de la acción de la masa en química, así como en sistemas bioquímicos y fisiológicos. Otros ejemplos de aplicación pueden ser a procesos económicos y a la producción de energía eléctrica.

El sistema de control BPID es básicamente una combinación de un controlador PID lineal estándar con una compensación bilineal. La adición de un compensador bilineal puede resultar en cierta linealización de una planta no lineal. Desde el punto de vista de un controlador, se consigue un sistema de control BPID ajustado.

Tras esto, el sistema linealizado se puede controlar mediante técnicas tradicionales como un controlador PID.

La estructura del controlador bilineal está representada en la figura 21.

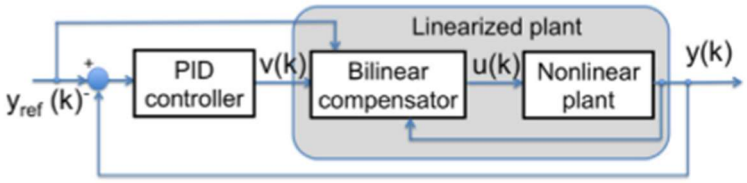


Figura 21. Representación esquemática de un controlador BPID [22]

2.2.4 Controladores PID basados en DE

Se ha desarrollado una nueva estrategia para estimar los parámetros de un controlador PID fraccionario que cumpla varias especificaciones para una planta. Este controlador depende del algoritmo DE, el cual se ha comentado anteriormente como una técnica de optimización evolucionaria basada en la minimización de una función objetivo. El diagrama de bloques del sistema de control se muestra en la figura 22.

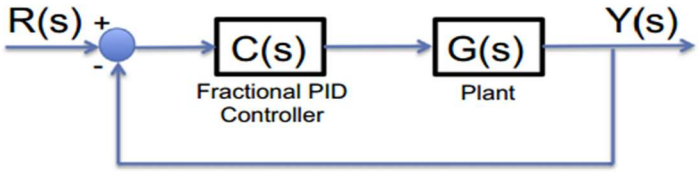


Figura 22. Representación esquemática de un controlador PID basado en DE [24]

Teniendo un proceso cuya función de transferencia G(s) es conocida, el algoritmo DE calcula los parámetros del controlador C(s) fraccionario para cumplir con las especificaciones en lazo cerrado.

La fórmula general de un controlador PID es

$$C(s) = k_p + \frac{k_i}{s^\lambda} + k_d s^\mu$$

donde λ y μ son los órdenes de las partes de integración y derivación del controlador, respectivamente. Las otras variables son las ganancias proporcional (k_p), integral (k_i) y derivativa (k_d) de un controlador PID clásico.

Se puede observar que hay dos parámetros más que ajustar para este tipo de controladores comparado con el PID tradicional. Esto significa que se pueden satisfacer más especificaciones con respecto a la robustez y el funcionamiento del sistema controlado.

Se puede usar este método de ajuste para cumplir con las especificaciones de control tanto en el dominio del tiempo como de la frecuencia. Por esta razón, se pueden usar dos funciones de coste diferentes., dependiendo de los objetivos que cumplir: la primera es una función de coste definida en el dominio del tiempo que minimiza el error entre la salida y la entrada de referencia; la segunda está definida para cumplir ciertos requisitos en el dominio de la frecuencia, desde la robustez a

las variaciones de ganancia de la planta.

El algoritmo DE y las funciones de coste se definen en [24] ajustadas para los controladores PID.

Ajuste en el dominio del tiempo

Una posibilidad de ajustar la función de coste a la hora de diseñar un controlador consiste en medir el error entre la salida y la entrada de referencia:

$$fitness(i) = \sum_{t=t_i}^{t_f} e_{TD}^2(t) = \sum_{t=t_i}^{t_f} [r(t) - y(t)]^2$$

donde $e_{TD}(t)$ es la señal de error en el dominio del tiempo. $r(t)$ es la señal de referencia (salida ideal que se desea obtener) y $y(t)$ es la salida real de la planta. El caso óptimo desde el punto de vista de control sería el caso en el que la señal de salida siguiera exactamente la señal de referencia.

Puesto que la señal está controlada en el dominio del tiempo, si el método de optimización resulta con éxito, se espera que las típicas características de control (tiempo de pico, tiempo de estabilización, sobreoscilación) presenten valores adecuados.

Sin embargo, hay muchos otros factores que no están incluidas en este tipo de control. Por ejemplo, que la robustez a las variaciones de la planta se ignore, que puede ocurrir en bastantes ocasiones en aplicaciones reales. Este factor sí se toma en cuenta al diseñar la función de coste en el dominio de la frecuencia.

Ajuste en el dominio de la frecuencia

Como se ha comentado anteriormente, una ventaja del ajuste en el dominio de la frecuencia es la capacidad de satisfacer más especificaciones incluidas en la función de coste.

Una de las especificaciones más comunes para controladores PID es la robustez a variaciones en la ganancia de la planta [25], que se puede obtener cumpliendo la siguiente condición:

$$\left.\frac{d(\arg(F(s)))}{d\omega}\right|_{\omega=\omega_{cg}} = 0$$

donde $F(s) = C(s)G(s)$ es la función de transferencia en lazo abierto y $G(s)$ es la función de transferencia de la planta.

La función de coste en este caso se calcula como la pendiente de la fase de la respuesta en lazo abierto sobre la frecuencia de cruce de ganancia.

$$fitness(i) = \sum_{\omega=\omega_i}^{\omega_f} e_{FD}^2(\omega) = (\varphi(\omega) - (\varphi_m - \pi))^2$$

donde el error se calcula en un intervalo entre ω_i y ω_f alrededor de la frecuencia de cruce de ganancia. $\varphi(\omega)$ es la fase del sistema en la frecuencia ω.

Se han incluido más requisitos en esta función de coste para cumplir con más especificaciones [23]:

- Intervalos para la frecuencia de cruce de ganancia y el margen de fase: Estas variables tienen una influencia importante en la robustez del sistema. Para calcular estos parámetros se siguen estas ecuaciones:

$$\left|C\left(j\omega_{cg}\right)G(j\omega_{cg})\right|_{dB} = 0\ dB$$

$$\arg\left(C\left(j\omega_{cg}\right)G\left(j\omega_{cg}\right)\right) = -\pi + \varphi_m$$

- Rechazo de ruido de alta frecuencia: El objetivo es reducir el ruido en altas frecuencias. La función de sensibilidad complementaria se calcula para cumplir esta restricción:

$$\left|T(j\omega) = \frac{C(j\omega)G(j\omega)}{1 + C(j\omega)G(j\omega)}\right|_{dB} = A\ dB$$

$$\forall\omega \leq \omega_t \text{ rad / s} \ \rightarrow\ |T(j\omega_t)|_{dB} \leq A\ dB$$

- Para rechazar perturbaciones de salida: Este requisito se cumple con la función de sensibilidad:

$$\left|S(j\omega) = \frac{1}{1 + C(j\omega)G(j\omega)}\right|_{dB} = B\ dB$$

$$\forall\omega \leq \omega_s \text{ rad / s} \ \rightarrow\ |S(j\omega_s)|_{dB} \leq B\ dB$$

Capítulo 3. El algoritmo RRA

En este capítulo se detalla la explicación matemática del algoritmo propuesto, su funcionamiento general y desarrollo a lo largo de las iteraciones, así como un pseudo-código simple que muestre de manera visual su configuración.

Se divide en dos secciones:

En la Sección 3.1 se recogen las explicaciones matemáticas del algoritmo, las ecuaciones a partir de las cuales se desarrolla tanto la inicialización como los procesos internos del algoritmo. Finalmente se muestra de manera esquemática en una figura su modo de operación.

En la Sección 3.2 se muestra una tabla que contiene el pseudo-código del RRA con comentarios explicativos de cada proceso.

3.1 Concepto general del algoritmo RRA

Al igual que otros algoritmos de optimización metaheurísticos, el algoritmo propuesto empieza con una población inicial aleatoria uniformemente distribuida en el dominio del problema, cada una denominada planta madre. El número de plantas madres se considera igual a N_{pop}. Después, en cada iteración, cada planta madre, excepto la más apta, genera una planta hija (utilizando tallos) aleatoriamente en el dominio del problema (búsqueda global de mejores soluciones). La distancia que cada una de las hijas con su madre correspondiente es controlada por un parámetro constante llamado d_{runn} (cuanto mayor sea d_{runner} mayor será la distancia entre una planta hija y su madre).

La planta madre más apta genera una planta hija exactamente en la misma ubicación que ella misma. Siguiendo este procedimiento se generan N_{pop} plantas hijas. Si al menos una de esas hijas conlleva una mejora considerable en el valor de la función coste comparado con el valor obtenido al principio del algoritmo, todas las plantas madres de la siguiente iteración son seleccionadas a partir de las plantas hijas utilizando una combinación de los métodos de selección de élite y de la rueda de ruleta. En concreto, en este caso una de las N_{pop} plantas madres requeridas para la siguiente iteración se considera igual a la planta hija más apta de la presente iteración (selección de élite) y el resto se obtienen aplicando el método de selección por rueda de ruleta a estas plantas hijas.

Esta sentencia es equivalente al hecho de que mientras se observe una mejora considerable en el valor de la función coste, el algoritmo investiga continuamente en el dominio del problema mejores soluciones sólo a través de tallos (búsqueda global) que se desarrollan en las aproximaciones de las soluciones actuales (plantas madres de la presente iteración). La amplitud de esta mejora se mide con el parámetro llamado "tol" el cual es equivalente al cambio relativo (o absoluto) en los mínimos valores obtenidos en la función coste de dos iteraciones sucesivas para todas las hijas.

Pero si ninguno de estas resultantes plantas hijas conlleva una mejora considerable para la función coste, el algoritmo comienza con el procedimiento de

la búsqueda local por medio de raíces y pelos radicales. En este caso el algoritmo asume que la planta hija más apta (entre aquellas obtenidas en la iteración actual) está ubicada exactamente en el mismo valle que la mejor solución global desconocida, y consecuentemente, aplicando un cambio aleatorio a cada variable de esa planta hija por separado y aceptando dicho cambio en caso de observar cualquier mejora se puede mover hacia la mejor solución global.

En concreto, si el cambio aleatorio aplicado a una cierta variable de la planta hija más apta conlleva una mejor solución, el algoritmo acepta dicho cambio y aplica otro cambio aleatorio a otra variable de la planta hija resultante anteriormente. Además, el anterior valor de dicha variable se mantiene y se aplica otro cambio aleatorio para la siguiente variable de la misma planta hija.

Esta rutina se aplica a todas las variables de la planta hija más apta una por una (nótese que, durante el proceso de búsqueda global, se han aplicado simultáneamente cambios aleatorios a todas las variables de las plantas madres seleccionadas mientras que durante la búsqueda local sólo las variables de la planta hija más apta son sometidas a cambios aleatorios una por una).

Claramente, esta estrategia asegura el movimiento general de la planta más apta hacia el punto mínimo más cercano ubicado en el mismo valle, el cual puede ser la mejor solución global. Como en este caso puede ocurrir que la planta más apta se ubique en un valle muy plano o estrecho, tanto los pequeños como los amplios cambios aleatorios se deberían aplicar a cada variable de la mejor planta hija. Estos cambios amplios y cortos, que asumen el rol de raíces y pelos radicales en la naturaleza, son aplicados por separado y controlados por los parámetros constantes d_{runner} y d_{root}, respectivamente.

Tras haber procedido con las búsquedas global y local (si fuese necesario) que se han mencionado anteriormente, las plantas madres de la siguiente iteración son seleccionadas de entre las plantas hijas resultantes por el método de selección de élite y selección por rueda de ruleta (nótese que de acuerdo a la descripción anterior, en cada iteración sólo una de las plantas hijas, que resulta ser la más apta antes de aplicar la búsqueda local, es sometida al ajuste fino mediante este procedimiento de búsqueda local, mientras que el resto de plantas hijas se mantiene sin cambios).

Para resumir, en cada iteración primero se realiza una búsqueda global y se genera una planta hija por cada planta madre. Si ninguna de estas plantas hijas conlleva una mejora considerable en el valor de la función coste, las variables de la planta hija resultante más apta son sometidas a cambios aleatorios una por una con ambos largos y cortos pasos.

El hecho de que la búsqueda local no se aplique en todas las iteraciones ni a todas las plantas hijas puede reducir considerablemente el número de evaluaciones de función. Sin embargo, ya que puede ocurrir que finalmente el algoritmo quede atrapado en un punto óptimo local, si la mejoría relativa en el valor de la función coste en dos iteraciones consecutivas es menor a "tol" (descrito anteriormente) el

valor de un contador denominado "$stall\_cou$ " aumenta en uno, si no, se mantiene igual a cero. Si el valor de este contador llega al valor predefinido igual a 2, el algoritmo se reinicia con una nueva población aleatoria inicial.

En este caso, todas las soluciones obtenidas anteriormente son descartadas excepto la mejor solución obtenida en la última iteración para determinar la mejor solución final (es decir, cuando el algoritmo se reinicie, ninguna solución, excepto la mejor, se transfiere desde la última iteración lo que conlleva a una re-inicialización en la próxima iteración). Hay que tener en cuenta que cada vez que el RRA se usa para resolver un problema de optimización puede resultar que se reinicie más de una vez o que no se reinicie en ningún caso. En los siguientes párrafos se estudian las diferentes partes del algoritmo en más detalle.

Considerando el siguiente problema de optimización sin restricciones:

$$min\, f(x), \qquad Xl \leq x \leq Xu \quad (1)$$

donde f: $Rm \rightarrow R$ es la función de coste de m variables a minimizar, el vector de la mejor solución global a calcular es $x^* = arg\, min\, f(x) \in Rm$, y $Xl,\ Xu \in Rm$ son dos vectores indicando los valores máximos y mínimos de las variables.

En caso de tratar con problemas de optimización restringidos se pueden usar métodos estándar para convertir dichos problemas en el equivalente sin restricciones [26].

Como se ha mencionado anteriormente, el RRA empieza con un conjunto de $Npop$ vectores generados aleatoriamente de dimensión m en el dominio del problema cada uno actuando como una planta madre. Después, en cada iteración todas las plantas madres excepto la mejor de ellas generan una planta hija aleatoria mientras que la mejor planta madre genera una hija exactamente igual a ella misma.

Siendo $X^k_{mother}(i)$ la ubicación de la $k^{\underline{o}}$ planta madre $(k = 1, \dots, N_{pop})$ en la iteración $i^{\underline{o}}$ donde $X^1_{mother}(i)$ es igual a la planta hija más apta de la anterior iteración (selección elitista) y $X^k_{mother}(i)$ $(k = 2, \dots, N_{pop})$ son el resto de plantas madres seleccionadas a partir de las plantas hijas anteriores mediante rueda de ruleta.

Es decir, usando esta notación, la ubicación de la $k^{\underline{o}}$ planta hija en la $i^{\underline{o}}$ iteración, $X^k_{daughter}(i)$, se calcula como:

$$X^k_{daughter}(i) = \begin{cases} X^1_{mother}(i), & k = 1 \\ X^k_{mothe}\ (i) + d_{runner} * r_k & k = 2, \dots, N_{pop} \end{cases} \quad (2)$$

donde $r_k \in Rm$ es un vector cuyas entradas son números aleatorios independientes de distribución uniforme en el rango [-0.5 , 0.5], y d_{runner} es un escalar que representa la distancia máxima entre las plantas hijas y la planta madre.

De acuerdo a (2) la mejor planta hija de la última iteración se considera necesariamente como una planta madre, así como una planta hija, en la presente iteración. Para calcular las plantas hijas se necesita crear un nuevo vector r_k.

El valor que se le asigna a d_{runner} tiene que ser lo suficientemente grande como para proveer a las plantas madres atrapadas en un mínimo local de una posibilidad de escapar y seguir buscando a la mejor solución global (la cual debe estar en un punto tan lejano como $Xu - Xl$ desde el punto mínimo local actual). Es por ello que los tallos juegan un papel muy importante para saltar sobre los mínimos locales, lo que ayuda efectivamente al algoritmo a evitar atraparse en esos puntos. Naturalmente, al tratarse de un problema de optimización de límites conocidos de variables como se comentaba en (1) el valor de d_{runner} tiene que ser comparable con el valor más grande de $Xu - Xl$.

La función de coste se evalúa en las ubicaciones calculadas en (2). Si al menos una de estas plantas hijas conlleva una mejora significante en el valor de la función de coste comparado con el de la mejor planta hija de la iteración anterior, es decir, si la desigualdad (3) se cumple,

$$\left| \frac{\min\limits_{k=1,\dots,N_{pop}} f\left(X^k_{daughter}(i)\right) - \min\limits_{k=1,\dots,N_{pop}} f\left(X^k_{daughter}(i-1)\right)}{\min\limits_{k=1,\dots,N_{pop}} f\left(X^k_{daughter}(i-1)\right)} \right| \geq tol \quad (3)$$

el algoritmo no comienza con la búsqueda local ya que (3) significa que la búsqueda global ha resultado efectiva (realizar la búsqueda local cuando la global ha encontrado una solución mejor no es una buena opción ya que consume bastantes evaluaciones de función). Si la función de coste se define como el denominador de (3) con posibilidad de ser igual a cero, la diferencia absoluta se puede utilizar en vez del término de la parte izquierda de la ecuación (3).

Pero si (3) no se cumple, debería ejecutarse la búsqueda local. Para este propósito se define a la planta hija más apta de aquellas calculadas en (2) como $X_{daughter,best}(i)$, y al vector obtenido al aplicar un cambio aleatorio a la kº entrada de $X_{daughter,best}(i)$ como $X_{perturbed,k}(i)$, es decir:

$$X_{perturbed,k} = diag\,\{1, 1, \dots, 1, 1 + d_{runner} * n_k, 1, \dots, 1\} * X_{daughter,best}(i) \quad (4)$$

donde d_{runner} es el mismo que antes, n_k $(k = 1, \dots, m)$ es un número aleatorio de distribución normal $(\mu = 0\ y\ \sigma = 0)$ y $diag\,\{1, 1, \dots, 1, 1 + d_{runner} * n_k, 1, \dots, 1\}$ es la matriz diagonal cuyos elementos de la diagonal son igual a 1 excepto el kº que es igual a $1 + d_{runner} * n_k$.

Para proceder con la búsqueda local, se calcula $X_{perturbed,k}$ para $k = 1$ a partir de (4). Si $f(X_{perturbed,1})$ < $f(X_{daughter,best}(i))$ entonces $X_{daughter,best}(i) \leftarrow X_{perturbed,1}$, si no, $X_{daughter,best}(i)$ permanece sin cambios. Este proceso se repite para todas las m variables de $X_{daughter,best}(i)$.

Hasta aquí se ejecuta la búsqueda local con pasos amplios (modelo de función basado en las raíces de las plantas en la naturaleza). Ahora, se realiza la

búsqueda local con pequeños pasos (modelo de función basado en los pelos radicales de las raíces en la naturaleza). Esta tarea se ejecuta de forma similar a la búsqueda local con amplios pasos descrita anteriormente y con la única diferencia de usar la siguiente ecuación en vez de la ecuación (4):

$$X_{perturbed,k} = diag\{1, 1, \dots, 1, 1 + d_{root} * r_k, 1, \dots, 1\} * X_{daughter,best}(i) \quad (5)$$

donde d_{root} es una constante escalar considerablemente más pequeña que drunner, $X_{daughter,best}(i)$ es el resultado final de la búsqueda local con amplios pasos realizada antes, r_k $(k = 1, \dots, m)$ es un número aleatorio de distribución uniforme en el rango [-0.5 , 0.5].

Nótese que para ninguna de las ecuaciones (4) y (5) se necesitan realizar operaciones de matrices, sino que simplemente significan que únicamente la entrada $k^{\underline{o}}$ de $X_{daughter,best}(i)$ es sometida a un cambio aleatorio.

Tras haber realizado la búsqueda local (en caso de ser necesario) se seleccionan las plantas madres para la siguiente iteración entre las plantas hijas de la presente iteración usando una combinación de técnicas de selección, la selección elitista y la selección por rueda de ruleta. La selección elitista se realiza de la siguiente forma:

$$X_{mot}^1 \quad (i+1) \leftarrow X_{daughter,best}(i) \quad (6)$$

Antes de aplicar la técnica de selección por rueda de ruleta para seleccionar al resto de plantas madres de la siguiente iteración, se calcula la aptitud de la $k^{\underline{o}}$ planta hija $(k = 1, \dots, Npop)$ de la siguiente manera:

$$fit\left(X_{daughter}^k(i)\right) = \frac{1}{a + f\left(X_{daughter}^k(i)\right) - f(X_{daughter,best}(i))} \quad (7)$$

donde a es una constante real positiva que controla la presión de selección. De acuerdo a (7) la planta hija más apta consigue el máximo valor de aptitud igual a $1/a$ y el resto de plantas hijas reciben valor de aptitud más pequeños.

Tras calcular estos valores, se calcula la probabilidad de que la $k^{\underline{o}}$ planta hija de la presente iteración sea seleccionada como planta madre de la próxima iteración, p_k, con la siguiente ecuación:

$$p_k = \frac{fit\left(X_{daughter}^k(i)\right)}{\sum_{j=1}^{N_{pop}} fit\left(X_{daughter}^j(i)\right)} \quad (8)$$

Una gran mayoría de métodos (como por ejemplo que el método del ránking lineal usado en AG) se pueden usar para calcular la aptitud de las plantas hijas en vez de la ecuación (7).

En la figura 23 se muestra el procedimiento esquematizado del algoritmo para 3 iteraciones en una simulación simple. En la figura las flechas negras, rojas y azules se refieren a la primera, segunda y tercera iteración, respectivamente.

Primero se consideran los puntos y flechas rojos (la primera iteración). El punto x(0) es el punto inicial generado aleatoriamente en el dominio del problema.

Como se puede observar, primeramente parte un vector hacia un segundo punto aleatorio con el orden de d_{runner} pues se trata de la primera búsqueda local en el que la planta hija más apta lanza raíces en busca de zonas con mejores condiciones. En el caso efectivo de haber encontrado una mejora en esta búsqueda de amplios pasos, se procede como segundo paso del problema a la búsqueda local con pequeños pasos mediante el indicador d_{root}.

Nuevamente, se encuentra una mejoría en la aptitud de la planta hija y se procede a la selección de la planta madre por selección élite, que se corresponde con el punto x(1) pues nos encontramos en la segunda iteración. Una vez más vuelve a realizar el proceso de búsqueda local encontrando el punto azul como mejor resultado, que resulta ser también la planta madre para la siguiente iteración x(2).

El esquema se detiene aquí, a la espera de encontrar una mejora tanto en la búsqueda local de amplios pasos como en la de pequeños pasos (en caso de no haber encontrado un punto más apto usando amplios pasos).

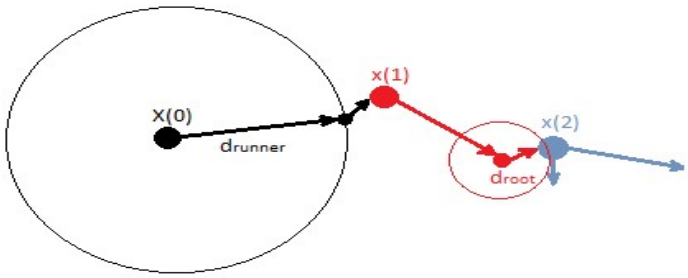


Figura 23. Representación esquemática del algoritmo RRA

3.2 Pseudo-código del algoritmo

El pseudo-código de RRA se presenta en la Tabla 1. En esta tabla, $rand$ representa un valor aleatorio con distribución uniforme en el rango [-0.5, 0.5], cada entrada de $r_k \in Rm$ se define igualmente como $rand$, y n_k $(k = 1, \ldots, m)$ es un número aleatorio con distribución normal $(\mu = 0\ y\ \sigma = 0)$.

Inicializar:

$$drunner, droot, Npop, stall\_max, tol, a$$

$$X^k{}_{mother}(1) := Xl + rand \cdot (Xu - Xl) \quad para\ k = 1, \dots, Npop$$

Búsqueda global:

$$X^k{}_{daughter}(i) = \begin{cases} X^1{}_{mother}(i), & k = 1 \\ X^k{}_{mother}(i) + d_{runner} \cdot r_1, & k = 2, \dots, Npop \end{cases}$$

$X_{daughter,best}(i) := \min f(x)$ {mínimo encontrado}

IF {mejora insuficiente}

Búsqueda local con amplios pasos:

$$Xperturbed, k := diag\{1, 1, \dots, 1, 1 + drunner \cdot nk, 1, \dots, 1\} \cdot Xdaughter, best(i)$$

$$Xdaughter, best(i) = Xperturbed, k$$ {En caso de mejora}

Búsqueda local con pequeños pasos:

$$Xperturbed, k := diag\{1, 1, \dots, 1, 1 + droot \cdot rk, 1, \dots, 1\} \cdot Xdaughter, best(i)$$

$$Xdaughter, best(i) = Xperturbed, k$$ {En caso de mejora}

END if

Selección elitista de la primera madre:

$$X^1 mother\ (i + 1) = Xdaughter, best(i)$$

Selección por rueda de ruleta del resto de madres:

$$chosen\_index = roullete(pk)$$

$$X^k mother(i + 1) = X^{ind} daughter(i)$$

IF {mejora insuficiente}

Posible condición de mínimo local

$$stall\_count = stall\_count + 1$$

END if

IF $stall\_count == 2$ {se confirma alcanzar un mínimo}

$$X^k{}_{mother}(i + 1) := Xl + rand \cdot (Xu - Xl)$$ {Reseteo de las madres}

END if

END

Tabla 1. Pseudo-código del algoritmo RRA

Capítulo 4. El controlador BPID

Durante este cuarto capítulo se realiza un estudio más exhaustivo del funcionamiento de los controladores bilineales introducidos anteriormente en el segundo capítulo.

4.1 Concepto general de un controlador BPID

Promovido por el hecho de que todos los sistemas prácticos exhiben un comportamiento no lineal, junto con el hecho de que las estructuras bilineales son capaces de modelar fenómenos no lineales con mayor precisión que las estructuras lineales se han diseñado estrategias de control bilineal para plantas industriales no lineales.

El controlador bilineal PID (BPID) que se describe en este documento, se considera que ofrece un compromiso realista entre el controlador PID estándar y otras alternativas más complejas [27], que han sido propuestas de manera similar por grupos de investigación académica durante la última década.

La investigación sobre los sistemas bilineales se remonta a la década de 1960, pero la intensidad de la actividad en esta área aumentó tras el gran avance realizado por Mohler a principios de los años setenta con su trabajo sobre modelización y control de la dinámica de los reactores nucleares [28].

En [29] se muestra que los sistemas bilineales exhiben un comportamiento dinámico dependiente del punto de funcionamiento y esto es representativo de una amplia gama de sistemas prácticos.

Los sistemas bilineales son considerados como un tipo específico de sistemas no lineales caracterizados por las siguientes ecuaciones:

$$\dot{x}(t) = Ax(t) + bu(t) + u(t)Nx(t) \qquad (9)$$

$$y(t) = c^T x(t) \qquad (10)$$

donde $x \in \Re^m$ es un vector de variables de estado y u, y, $\in \Re^m$ son las las variables de entrada de control y de salida del proceso, respectivamente. A es la matriz $n \times n$ de constantes reales, b es el vector $n \times 1$ de constantes reales, N es la matriz $n \times n$ de constantes reales, que comprende los coeficientes bilineales, y c es el vector de salida $n \times 1$ de constantes reales. Las matrices se definen de la siguiente manera:

$$A = \begin{bmatrix} 0 & 1 & . & . & 0 & 0 \\ 0 & 0 & . & . & 0 & 0 \\ . & . & . & . & . & . \\ . & . & . & . & 1 & 0 \\ 0 & 0 & . & . & 0 & 1 \\ -\alpha_0 & -\alpha_1 & . & . & -\alpha_{n-2} & -\alpha_{n-1} \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 0 \\ 1 \end{bmatrix}, \qquad (11)$$

$$N = \begin{bmatrix} 0 & 1 & . & . & 0 & 0 \\ 0 & 0 & . & . & 0 & 0 \\ . & . & . & . & . & . \\ . & . & . & . & 1 & 0 \\ 0 & 0 & . & . & 0 & 1 \\ \rho_0 & \rho_1 & . & . & \rho_{n-2} & \rho_{n-1} \end{bmatrix}, \qquad c^T = [\beta_0 \quad \beta_m \quad 0 \quad 0] \qquad (12)$$

Combinando estos términos con las ecuaciones de estado (9) y (10) un sistema SISO bilineal se puede expresar como:

$$\dot{x}(t) = [A + u(t)N]x(t) + bu(t) \qquad (13)$$

donde se hace evidente que las características de respuesta dinámica y en régimen permanente dependen de la entrada.

De hecho, para cualquier sistema bilineal de la forma (9) y (10), la salida en régimen permanente Y_{ss}; (El subíndice ss denota el valor de régimen permanente *"steady-state"*) correspondiente a una entrada en régimen permanente U_{ss} está dada por:

$$Y_{ss} = \frac{\beta_0 U_{ss}}{\alpha_0 - \rho_0 U_{ss}} \qquad (14)$$

Las características de las entradas y salidas en régimen permanente para los tres casos diferentes del término bilineal se ilustran en la Figura 24 [30]:

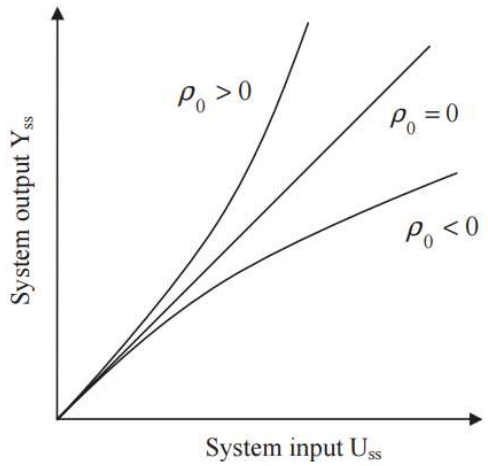


Figura 24. Ilustración de las características de las entradas y salidas en estado estacionario de un sistema bilineal

Claramente, si ρ_0 es cero, la ecuación (14) representa un sistema lineal, por lo que los sistemas lineales pueden ser considerados como una subclase especial. Una consecuencia de esta estrecha relación entre sistemas lineales y bilineales es que muchas técnicas desarrolladas para sistemas lineales pueden extenderse y aplicarse a casos bilineales.

Valores positivos de ρ_0 resultan en una ganancia que aumenta a medida que U_{ss} aumenta, típico de los procesos químicos exotérmicos. Por el contrario, ρ_0

negativo produce una ganancia que disminuye a medida que U_{ss} aumenta, dando lugar a una consiguiente saturación, típico de muchos sistemas industriales. Si un sistema presenta características bilineales de la forma ilustrada en la Figura 24, entonces es oportuno considerar el enfoque de modelos y control de sistemas bilineales.

La estructura de este tipo de controladores tiene además otra ventaja, pues el orden de un sistema bilineal es menor que su correspondiente modelo lineal. Martineau concluyó [31] que la siguiente solución podía aplicarse a un modelo de aplicaciones de un horno industrial.

$$y(k) = -a \;\; y(k-1) + b \;\; u(k-1) + \eta \;\; u(k-1)y(k-1) \qquad (15)$$

donde $y(k)$ representa la salida de la planta, $u(k-1)$ es la entrada discreta y a, b, y η son constantes. Reorganizando la ecuación, se obtiene:

$$y(k) = -a \;\; y(k-1) + b \;\; \left[1 + \frac{\eta}{b} y(k-1)\right] u(k-1) \qquad (16)$$

Este tipo de sistemas no lineales se pueden controlar mediante una técnica basada en añadir una nueva entrada v. La relación entre la nueva entrada y la original es:

$$u(k-1) = \frac{1}{1 + K_b y(k-1)} v(k-1) \qquad (17)$$

donde $K_b = \eta/b$ es el parámetro de ajuste.

Al sustituir (4) en (3) se observa que el sistema es ahora lineal:

$$y(k) = -a \;\; y(k-1) + b \;\; v(k-1) \qquad (18)$$

Tras esto, el sistema linealizado se puede controlar mediante técnicas tradicionales como un controlador PID.

El último componente que se añade al compensador bilineal es un término que garantiza una ejecución correcta mantenida en el punto de ajuste. Finalmente la fórmula del compensador propuesta por Martineau es:

$$\frac{1 + K_b y_{ref}(k)}{1 + K_b y(k-1)} \qquad (19)$$

donde $y_{ref}(k)$ es la salida de referencia en el que el controlador PID está ajustado.

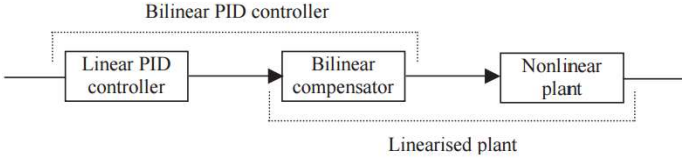


Figura 25. Concepto básico de un controlador bilineal

Capítulo 5. Aplicación del algoritmo RRA a técnicas de control

En este capítulo se utiliza el algoritmo RRA como algoritmo de optimización para los distintos ejemplos de aplicación. Cada Sección corresponde a una aplicación distinta del algoritmo. En todas ellas, el software utilizado para la programación y simulación de los resultados es Matlab, una herramienta de software matemático con un lenguaje de programación propio [32]. Además, se utiliza una plataforma integrada en este software para la simulación multidominio denominada Simulink [33].

En la Sección 5.1 se trata de implementar el algoritmo y ajustar sus parámetros iniciales para conseguir una distancia mínima al punto. Con este ejercicio se pretende evaluar el comportamiento de los parámetros del algoritmo.

En la Sección 5.2 se pretende controlar la posición de un motor de corriente continua, aplicando el RRA para optimizar los valores de un controlador BPID.

Por último, en la Sección 5.3 se aplica el algoritmo propuesto para controlar mediante un BPID la velocidad de un motor de corriente continua, con una función de transferencia más compleja.

5.1 Control de la distancia mínima al punto

Para comprobar si el algoritmo resulta efectivo, se aplicará al caso más sencillo de calcular la distancia mínima al punto. El objetivo a minimizar es el punto [50 50 50].

Por ello entonces se trata de un problema con tres variables a optimizar, las cuales son el punto x, el punto y, y el punto z. Las plantas tendrán entonces 3 variables a las que someter a cambios aleatorios.

Para ello se preseleccionan los parámetros del algoritmo que mejor se adapten al dominio del problema.

En este caso se tienen los valores del domino del problema tal que:

- Xl=0
- Xu=100
- Npop=100
- stall_max=100

Lo que quiere decir que el límite inferior es 0 y el límite superior es 100, con una población de 100 plantas madres iniciales, evaluando el problema durante 100 iteraciones. Por tanto, definiremos las variables del algoritmo como:

- d_{runner}=30
- d_{root}=0.1
- tol=0.5
- a=0.1

La función coste de este problema vendría definida por la ecuación 20 de la distancia al punto:

$$coste = \sqrt{(x-50)^2 + (y-50)^2 + (z-50)^2} \qquad (20)$$

Por tanto, al proceder con la ejecución del algoritmo se puede observar en un primer momento el estado inicial del problema (Figura 26).

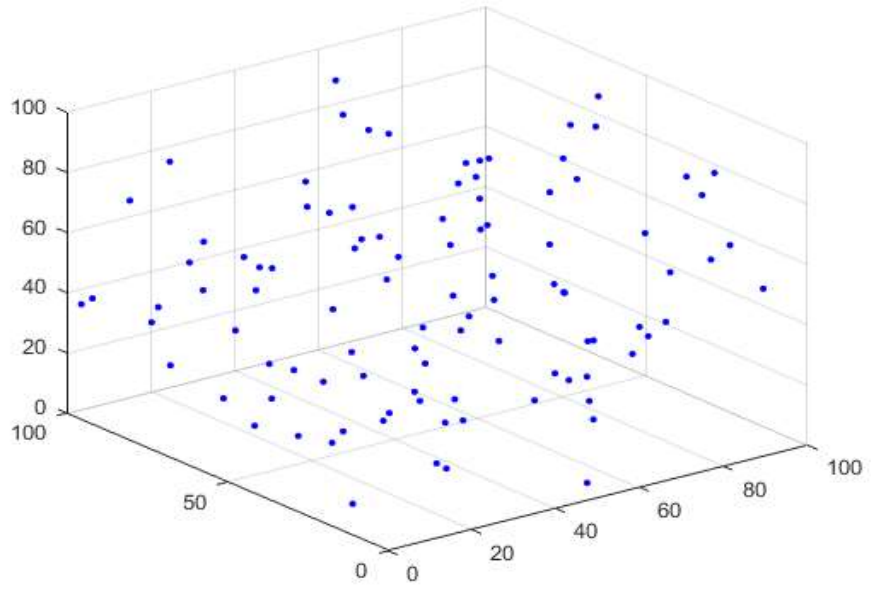
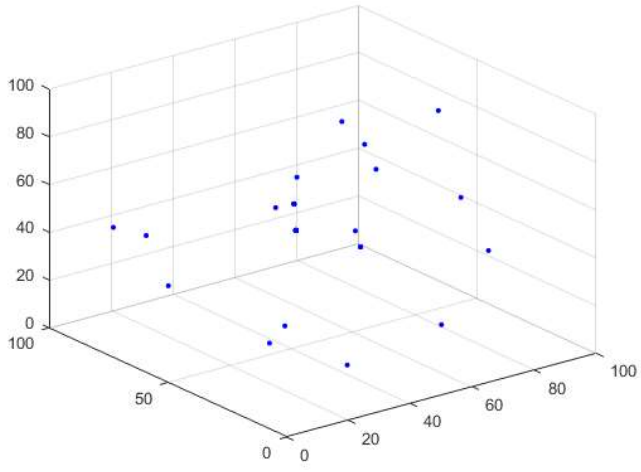


Figura 26. Población inicial (I)

Donde se pueden ver reflejadas las posiciones de las madres en el primer instante. Tras ello, se procede a la evaluación tanto de la búsqueda global de las hijas, y sucesivamente la selección de las madres por ruleta rusa para la próxima iteración.

Así pues, al finalizar la primera iteración se tiene la solución de la Figura 27.




```
>> RRA

bestmem =

   37.8367   45.3895   52.4819


error =

   13.2424


i =

     1
```

Figura 27. Población y mejor resultado tras la primera iteración (I)

Se puede observar en la representación que, tras efectuar el método de selección de las madres por ruleta rusa, se ha reducido el número de madres distintas aleatorias, pues muchas de ellas son idénticas.

Al continuar con el algoritmo durante una iteración más se puede ver su evolución en el resultado "error" pues debería disminuir con el paso de las iteraciones. Para la segunda iteración se obtiene la Figura 28.

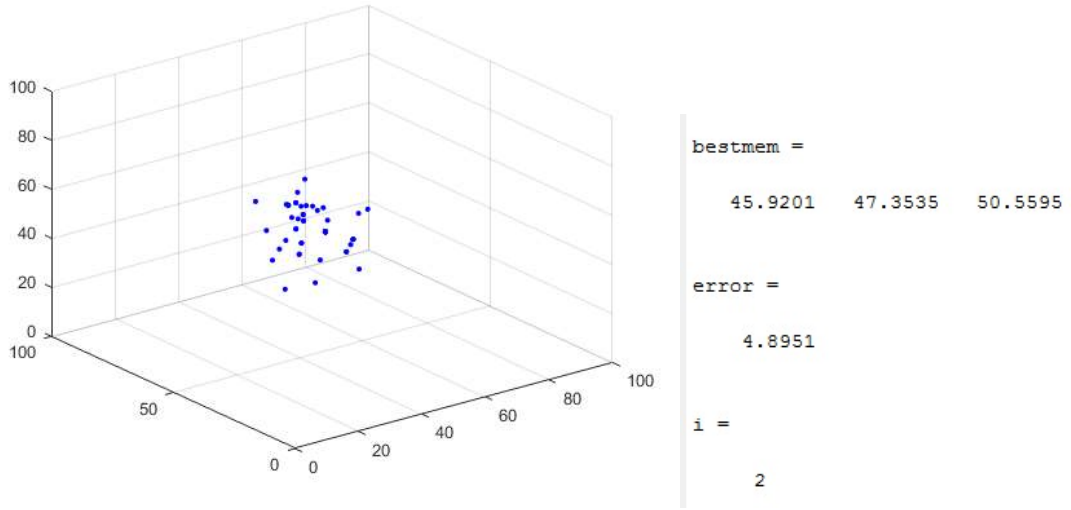


Figura 28. Población y mejor resultado tras la segunda iteración (I)

Claramente, se aprecia una predisposición de las plantas a situarse en torno al punto más óptimo, que en este caso se ha acercado más a la solución con un error de 4.8951 respecto al valor anterior de 13.2424.

Una vez realizadas todas las iteraciones, se obtienen los resultados de la Figura 29.

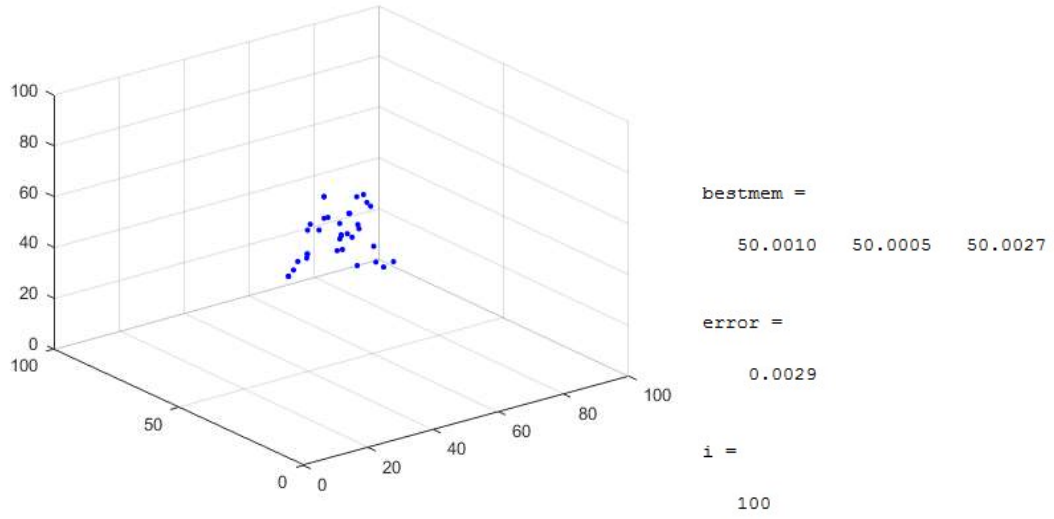


Figura 29. Población y mejor resultado tras finalizar la ejecución del algoritmo (I)

El error ha disminuido drásticamente obteniendo una solución más que válida para el problema planteado. En cualquier caso, gráficamente se observan a las plantas madres finales repartidas en un área bastante amplia. Ello puede ser debido al valor que se había preseleccionado en un primer momento de d_{runner}, pues podría haber resultado demasiado grande para una mejor representación.

Esta primera evaluación del problema permite ajustar los valores de las variables de inicialización más correctamente para futuras evaluaciones.

Se seleccionan ahora nuevos valores de inicialización:

- d_{runner}=5
- d_{root}=0.1
- tol=0.5
- a=0.1

Con los que se obtiene como población la correspondiente a la Figura 30.

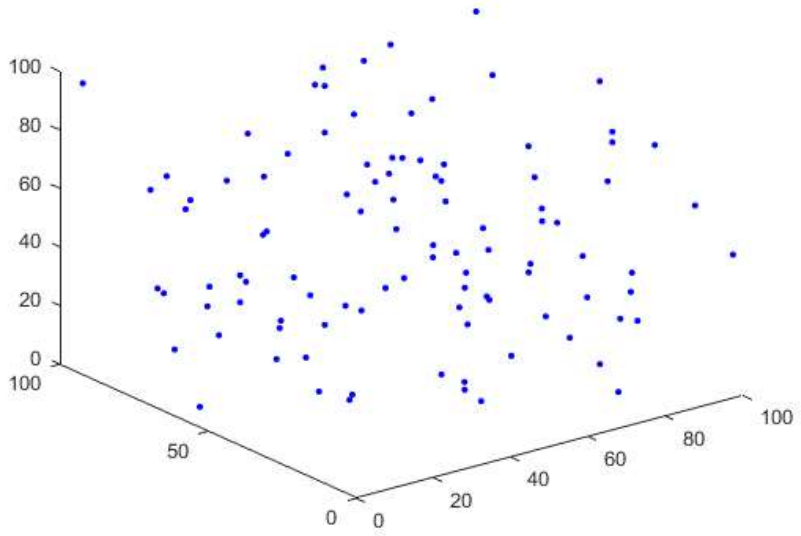


Figura 30. Población inicial (II)

Y tras la primera iteración la de la Figura 31.

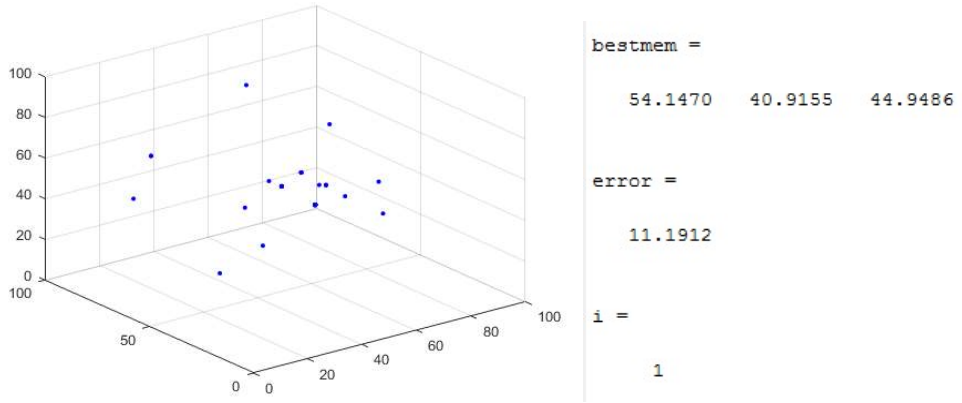


Figura 31. Población y mejor resultado tras la primera iteración (II)

Se pueden ver los puntos de la primera selección menos repartidos que en el caso anterior.

Repitiendo el procedimiento hasta 100 iteraciones después, se obtiene la Figura 32.

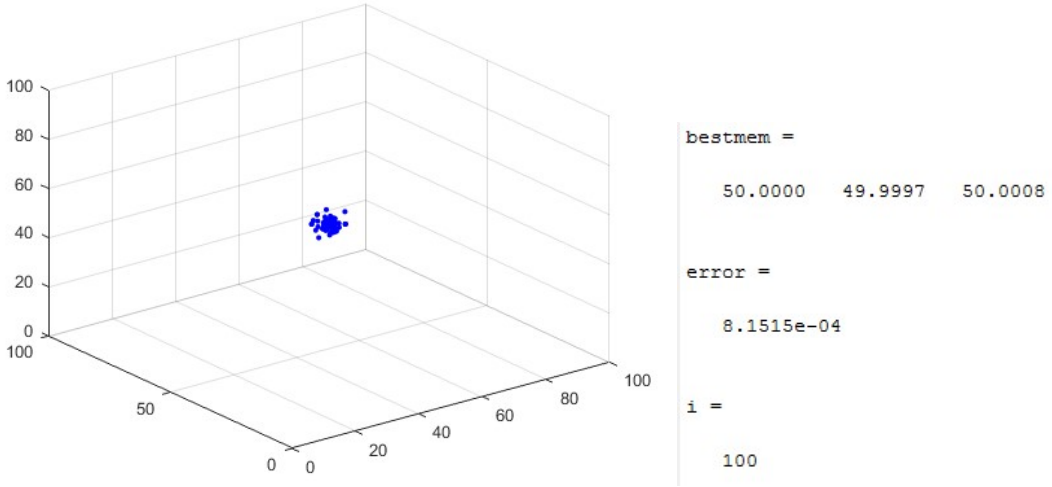


Figura 32. Población y mejor resultado tras finalizar la ejecución del algoritmo (II)

Los resultados ahora obtenidos resultan mucho más satisfactorios que los obtenidos en la primera evaluación. Al reducir la longitud de los tallos (runners) que determinan tanto la búsqueda global como la local en amplios pasos, la concentración de plantas madres aumenta en las vecindades de la planta más apta.

Para entender el funcionamiento de los otros parámetros de inicialización que son tol y a se realiza un estudio de ambos manteniendo el resto de parámetros constantes. En este caso, al haber encontrado una mejora en los parámetros drunner y droot, se mantendrán estos últimos pues han resultado más efectivos.

El parámetro tol se aplica en el algoritmo con la inecuación 21.

$$\left| \frac{\min\limits_{k=1,\dots,N_{pop}} f\left(X^k_{daughter}(i)\right) - \min\limits_{k=1,\dots,N_{pop}} f\left(X^k_{daughter}(i-1)\right)}{\min\limits_{k=1,\dots,N_{pop}} f\left(X^k_{daughter}(i-1)\right)} \right| \geq tol \qquad (21)$$

la cual representa que unos valores de tolerancia muy grandes supondrían un mayor consumo de evaluaciones del algoritmo, es decir, la condición para la búsqueda local del algoritmo se cumpliría con mayor facilidad, ejecutando este proceso de búsqueda en casos innecesarios en los que la búsqueda global ya ha resultado efectiva. Por otro lado, valores muy pequeños de tolerancia supondrían una reducción en la precisión del algoritmo pues el mínimo cambio en la búsqueda global resultaría lo suficientemente beneficioso para el algoritmo que no realizaría la búsqueda local.

El parámetro a, a su vez, se aplica en el algoritmo con la ecuación 22.

$$fit\left(X^k_{daughter}(i)\right) = \frac{1}{a + f\left(X^k_{daughter}(i)\right) - f\left(X_{daughter,best}(i)\right)} \qquad (22)$$

La cual representa la aptitud de las plantas hijas, influyendo en la probabilidad de ser elegidas como madres para la siguiente iteración. Por tanto, unos valores muy altos de a, aumentarían la probabilidad de las plantas hijas de ser elegidas como madres, lo cual no supondría un gran beneficio para el algoritmo pues está sobreviviendo una población que no es la más apta. Por otro lado, valores muy pequeños de este parámetro supondrían totalmente lo opuesto, otorgando menor probabilidad a las plantas hijas de mayor coste a ser elegidas como madres. En este caso, sólo las plantas hijas más aptas serían las elegidas como madres, reduciendo la población en gran número, lo que a su vez puede provocar que entre con mayor facilidad en un mínimo local, atrapando al algoritmo hasta cumplir la reinicialización.

En primer lugar, se comprueba cómo afecta la variación de tol al algoritmo. El resultado anterior se había conseguido con un valor de tol de 0,5. Para ver su comportamiento se aumenta su valor a 5, y se evalúan los resultados.

La población inicial resultante se representa en la Figura 33.

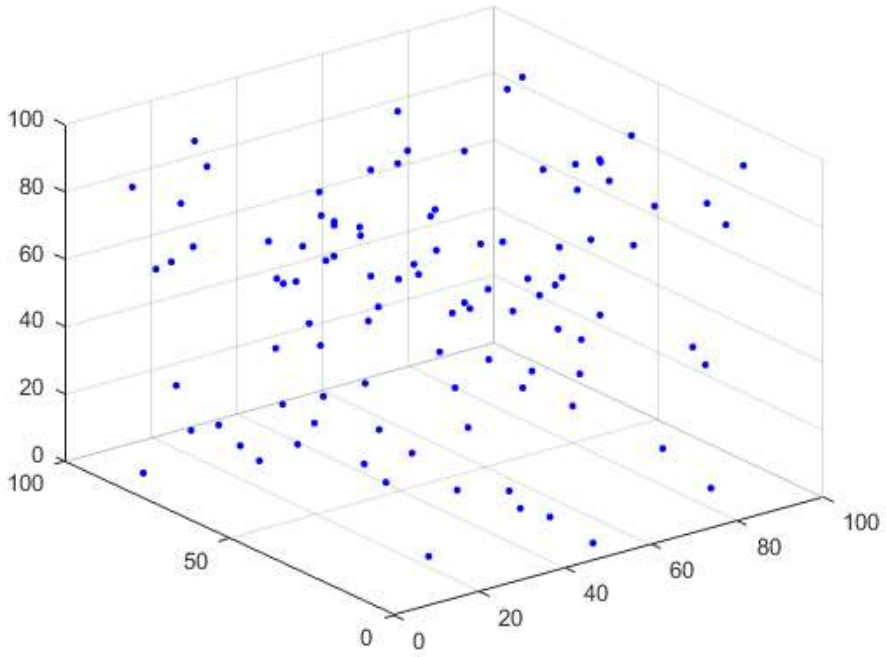


Figura 33. Población inicial (III)

Y ya en la segunda iteración se cumple la tolerancia, procediendo con la búsqueda local, como se muestra en la Figura 34.

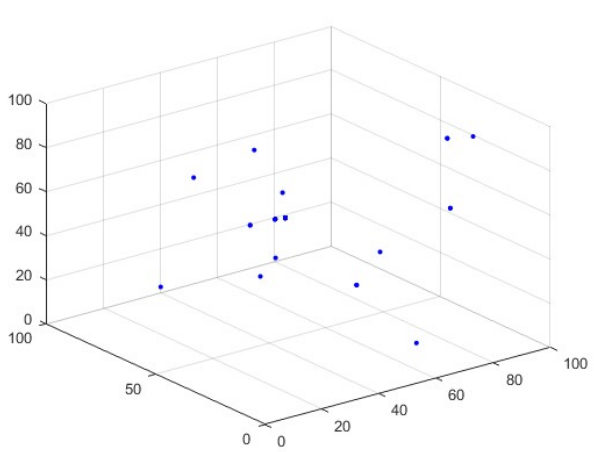


Figura 34. Población y mejor resultado tras la segunda iteración (III)

Por lo que consume muchos recursos cuando la búsqueda global aún resulta efectiva consumiendo menos evaluaciones de coste en cada iteración. Como resultado final se obtiene la Figura 35.

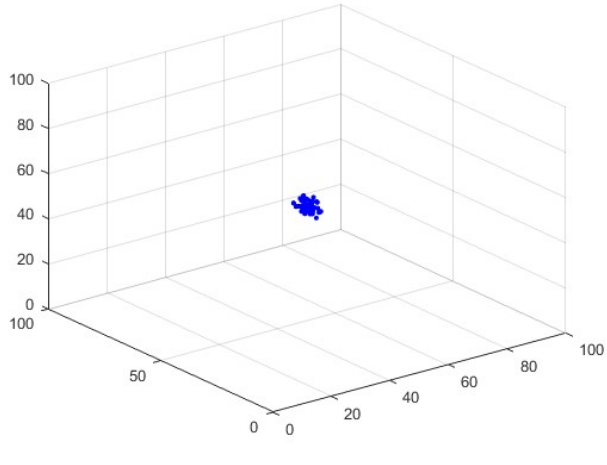


Figura 35. Población y mejor resultado tras finalizar la ejecución del algoritmo (III)

Sin embargo, reduciendo el valor de la tolerancia a 0,005 la situación es distinta, pues teóricamente se realizaría la búsqueda local del algoritmo cuando este esté más avanzado.

Para una población inicial como la de la Figura 36.

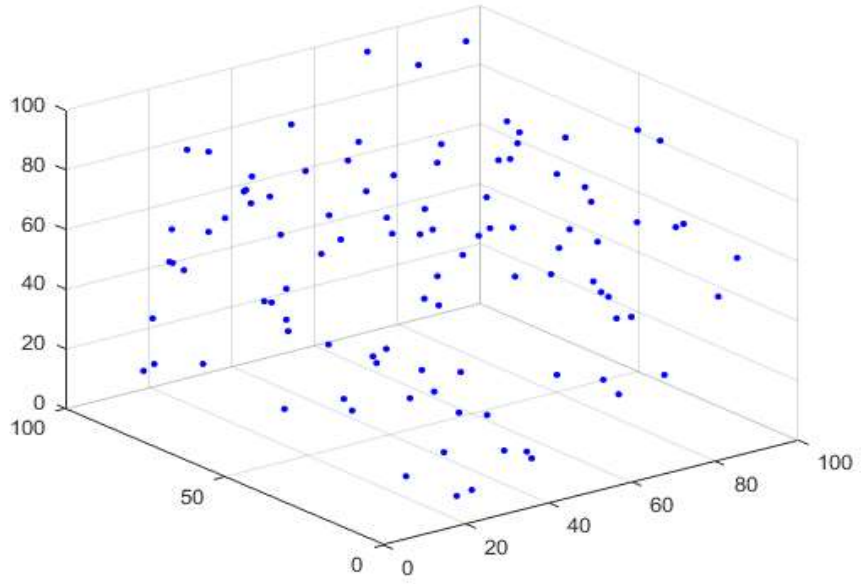
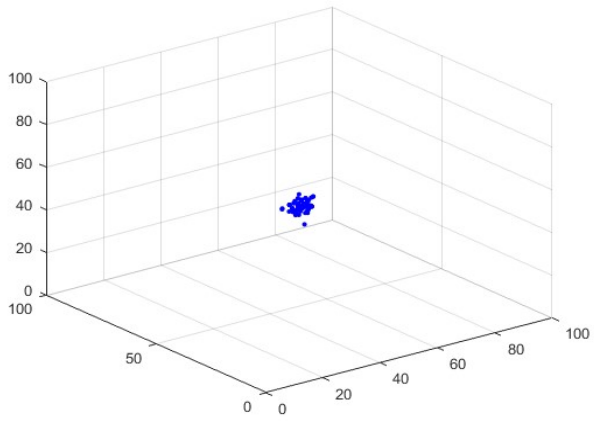


Figura 36. Población inicial (IV)

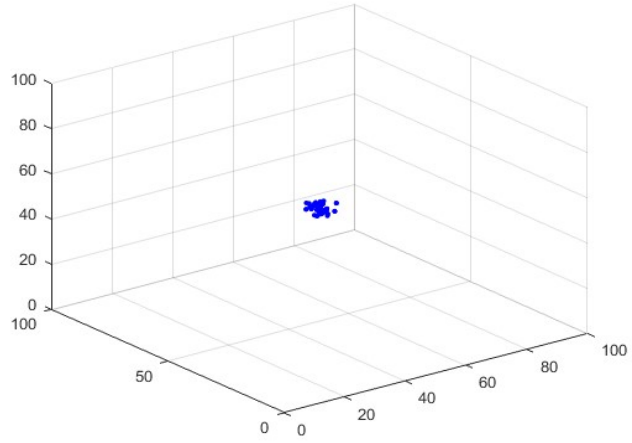
No se cumple la tolerancia hasta la séptima iteración donde la población queda representada en la Figura 37.




```
Tolerancia conseguida
bestmem =

   50.8524   50.0844   49.5252


error =

    0.9793


i =

     7
```

Figura 37. Población y mejor resultado tras la séptima iteración (IV)

Y obteniendo como resultado final la Figura 38.




```
Tolerancia conseguida
bestmem =

   50.0010   49.9995   50.0003


error =

    0.0012


i =

   100
```

Figura 38. Población y mejor resultado tras finalizar la ejecución del algoritmo (IV)

Por tanto, las predicciones en cuanto al comportamiento de este parámetro quedan satisfechas.

En cuanto al parámetro a, en las evaluaciones anteriores tenía un valor de 0,1. Aumentando su valor a 10 se puede observar su comportamiento para valores muy grandes. Además, el valor de tol vuelve a ser 0,5 para sólo contemplar la evolución del parámetro a.

Para la población inicial de la Figura 39.

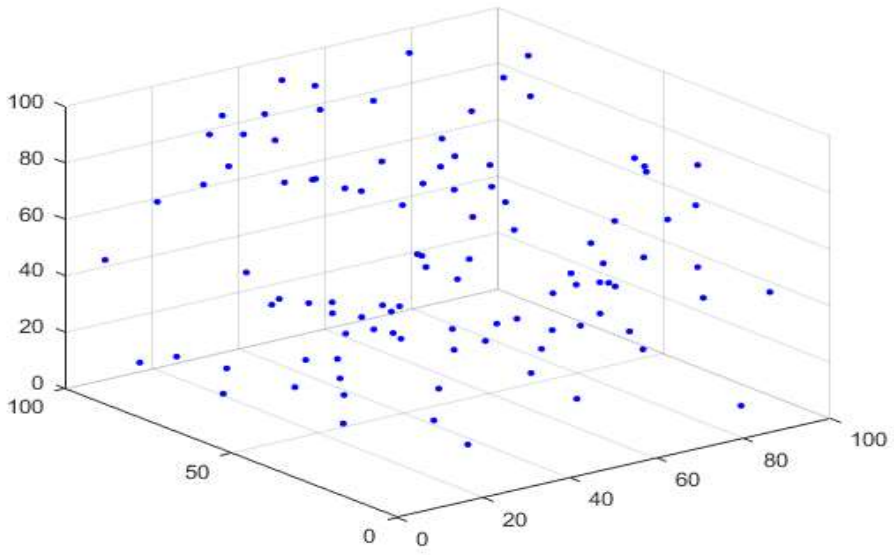
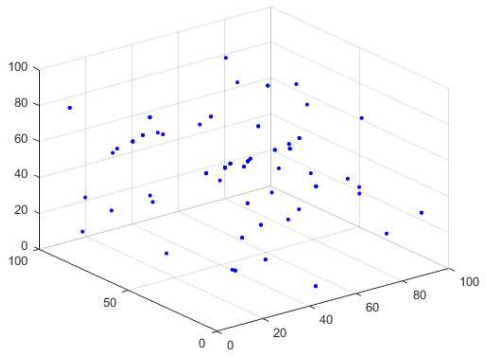
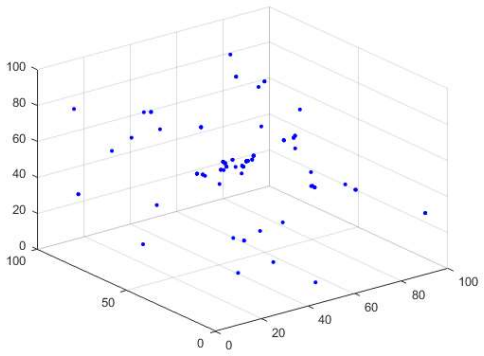


Figura 39. Población inicial (V)

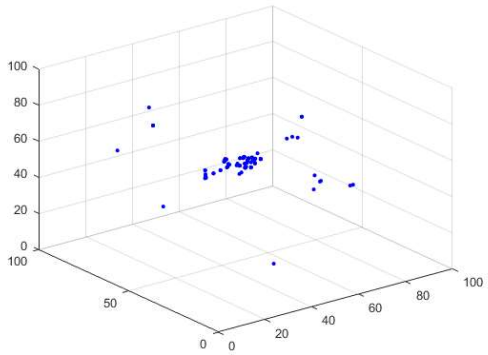
Evaluando las diez primeras iteraciones se puede observar en la Figura 40 cómo se convierten las plantas hijas en plantas madres, y cuáles consiguen sobrevivir con mayor probabilidad.



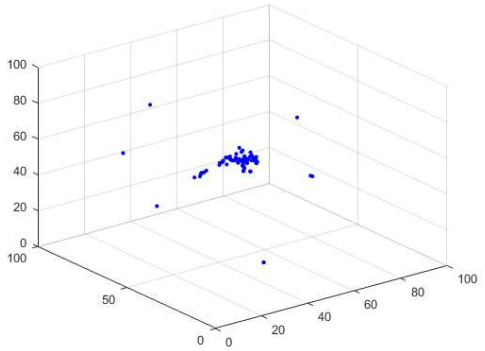
Iteración 1



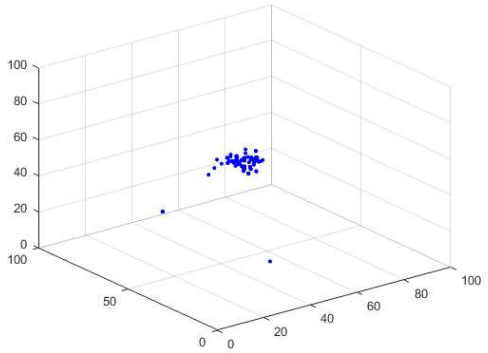
Iteración 2



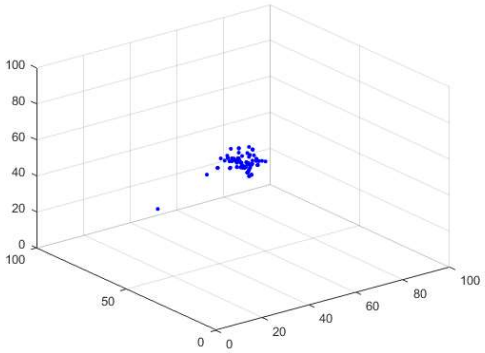
Iteración 3



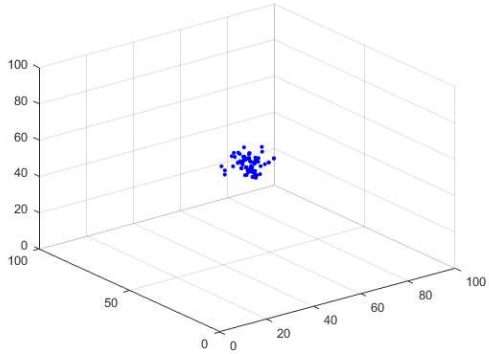
Iteración 4



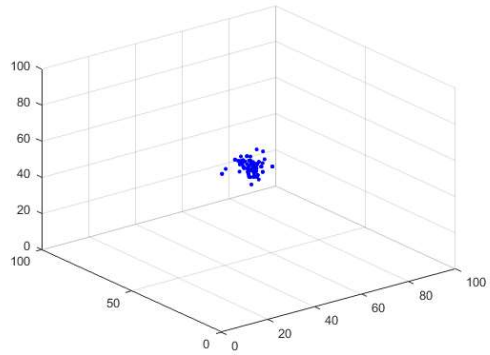
Iteración 5



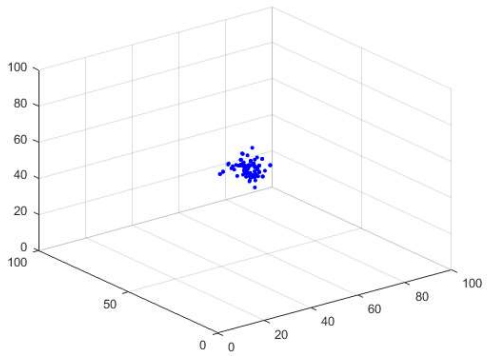
Iteración 6



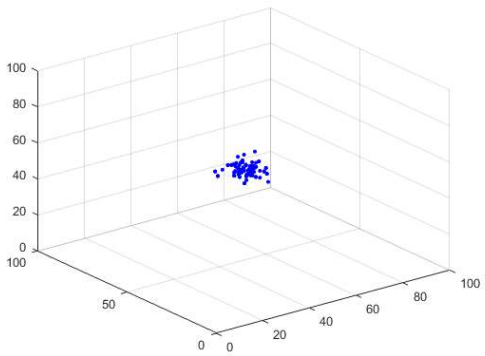
Iteración 7



Iteración 8



Iteración 9



Iteración 10

Figura 40. Evolución de la población durante diez iteraciones (V)

Como se puede observar, muchas plantas que no resultan ser de las más aptas, consiguen sobrevivir durante varias iteraciones. Mientras tanto, disminuyendo el valor del parámetro a, la población debería verse mucho más reducida en torno al punto más óptimo. Para ello, se evalúa de nuevo el problema con un valor de a de 0,001.

Para la población inicial de la Figura 41.

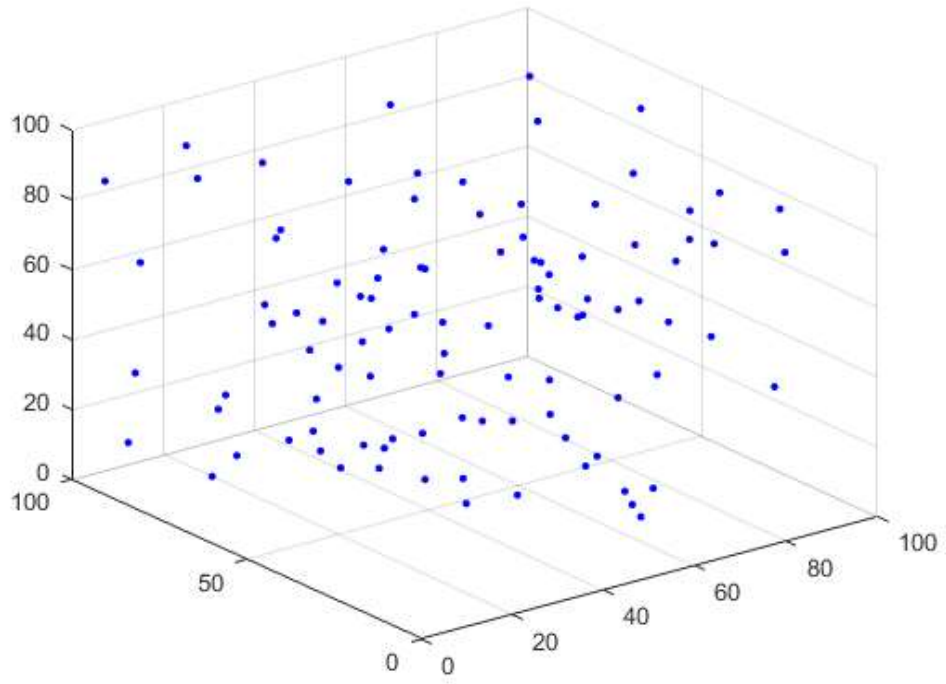
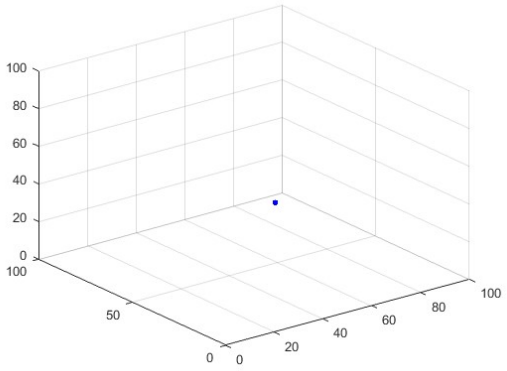
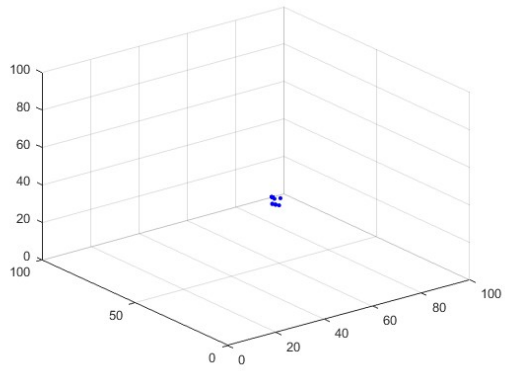


Figura 41. Población inicial (VI)

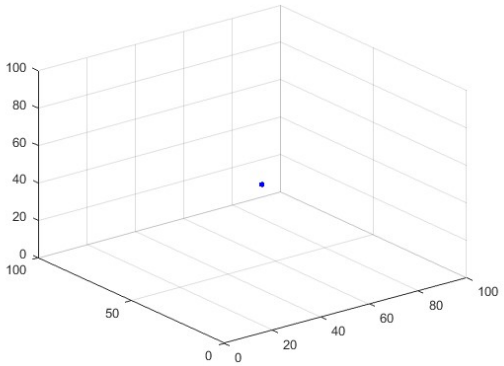
Se evalúan, en este caso, las diez primeras iteraciones con el resultado representado en la Figura 42.



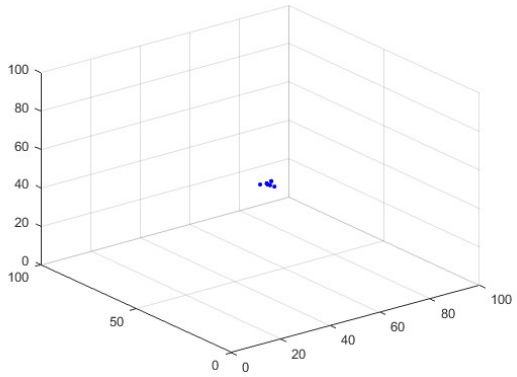
Iteración 1



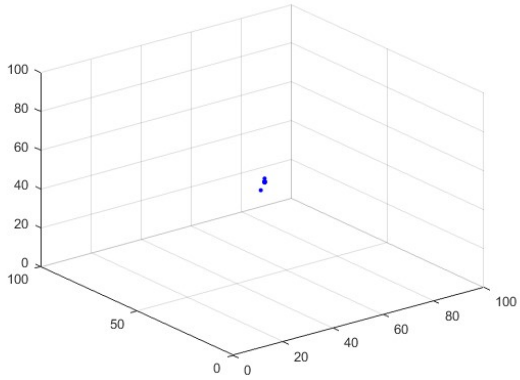
Iteración 2



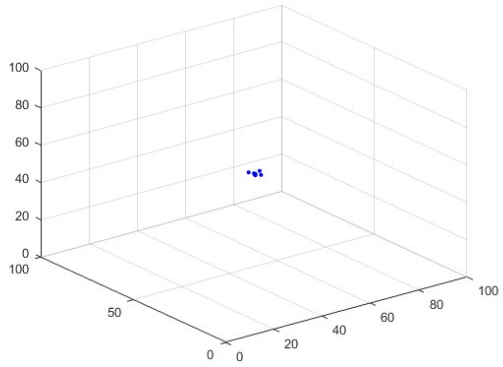
Iteración 3



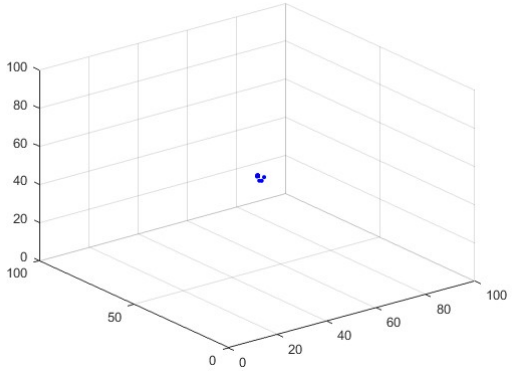
Iteración 4



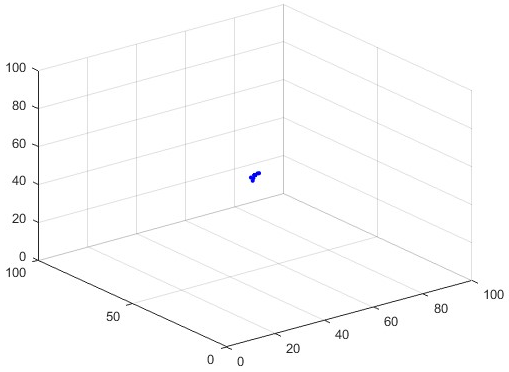
Iteración 5



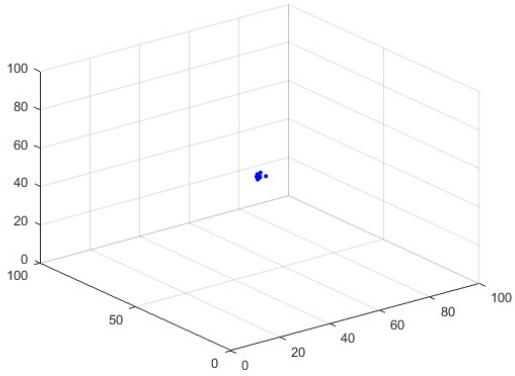
Iteración 6



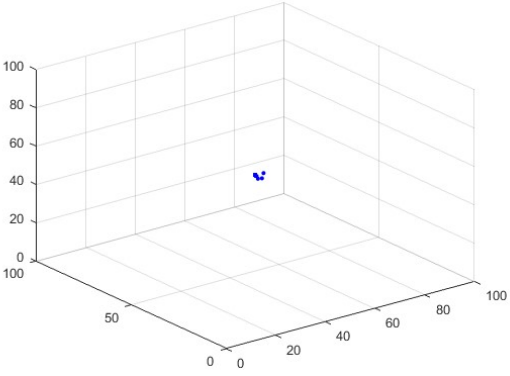
Iteración 7



Iteración 8



Iteración 9



Iteración 10

Figura 42. Evolución de la población durante diez iteraciones (VI)

El problema de la distancia al punto muestra claramente el funcionamiento del algoritmo, pues se estudia el comportamiento de las variables del algoritmo por separado. En ese apartado, se distinguen dos conjuntos de parámetros. Uno es el conjunto formado por los anteriormente comentados d_{runner} y d_{root}, los cuales influyen en el proceso de mutación del algoritmo. El otro conjunto es el formado por los parámetros a y tol los cuales influyen en mayor medida en el estado de reproducción del algoritmo, así como del tiempo de ejecución. Es por eso que una vez se han escogido unos valores aceptables para este problema, se mantienen en el resto del proyecto.

Se puede concluir con los resultados que el algoritmo trabaja de manera correcta y se puede aplicar a otros problemas de control más complejos.

5.2 Control de posición de un motor de corriente continua

Modelo del motor de corriente continua

Un actuador usado comúnmente en sistemas de control es el motor de corriente continua. Directamente produce un movimiento de rotación y, junto con unas ruedas y cables, puede producir movimiento traslacional. El circuito eléctrico equivalente al conjunto y el diagrama de cuerpo libre se representan con la Figura 43.

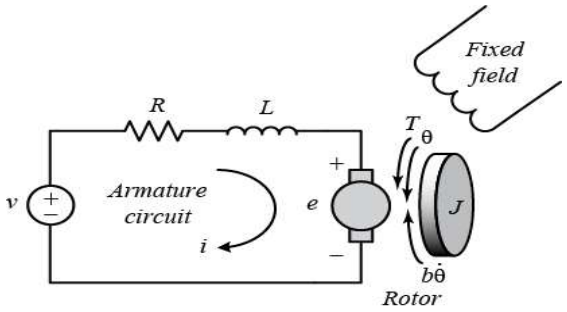


Figura 43. Esquema eléctrico y representación gráfica del modelo de motor en corriente continua

Para este ejemplo, se asumirán los siguientes valores para los parámetros físicos:

(J)	moment of inertia of the rotor	3.2284E-6 kg.m^2
(b)	motor viscous friction constant	$3.5077E-6\ N\cdot m\cdot s$
(Kb)	electromotive force constant	$0.0274\ \frac{V}{rad/s}$
(Kt)	motor torque constant	$0.0274\ \frac{N\cdot m}{Amp}$
(R)	electric resistance	$4\ Ohm$
(L)	electric inductance	$2.75E-6H$

Además, se asume que la entrada del sistema es una fuente de voltaje (V) aplicada a la armadura del motor, mientras que la salida del sistema es la posición del eje (θ). El eje y el rotor se suponen cuerpos rígidos. En añadido, se supone un par de fricción proporcional a la velocidad angular del eje.

Ecuaciones del sistema

En general, el par generado por un motor de corriente continua es proporcional a la corriente inducida y a la fuerza del campo magnético. En este ejemplo, se asume un campo magnético constante y, por lo tanto, que el par del motor es proporcional únicamente a la corriente inducida i por una constante Kt como se muestra en la ecuación 23:

(23) $$T = Kt \cdot i$$

La fuerza electromotriz, e, es proporcional a la velocidad angular del eje por un factor constante Kb:

(24) $$e = Kb \cdot \ddot{\theta}$$

En unidades del sistema internacional, las constantes del par motor y de la fuerza electromotriz son iguales, es decir, $Kt = Ke$; por lo que la constante K se usará para representar ambas.

Utilizando las ecuaciones de la segunda ley de Newton y las leyes de voltaje de Kirchhoff se obtienen las ecuaciones 25 y 26:

(25) $$J\ddot{\theta} + b\ddot{\theta} = Ki$$

(26) $$L\frac{di}{dt} + Ri = V - K\dot{\theta}$$

La transformada de Laplace se usa en este ejemplo para modelar las ecuaciones anteriores en términos de la variable s obteniendo las ecuaciones 27 y 28:

(27) $$s(Js + b)\Theta(s) = KI(s)$$

(28) $$(Ls + R)I(s) = V(s) - Ks\Theta(s)$$

Eliminando $I(s)$ se llega a una función de transferencia en bucle abierto (ecuación 29), donde la velocidad angular se considera la salida del sistema y el voltaje inducido la entrada del mismo.

(29) $$P(s) = \frac{\dot{\Theta}(s)}{V(s)} = \frac{K}{s((Js+b)(Ls+R)+K^2)}$$

Sin embargo, en este ejemplo se pretende observar la posición del motor, por lo que hay que transformar la ecuación anterior integrando la velocidad, es decir, dividiendo la función de transferencia anterior por s, como en la ecuación 30:

(30) $$\frac{\Theta(s)}{V(s)} = \frac{K}{s((Js+b)(Ls+R)+K^2)}$$

Aplicando los valores predefinidos anteriormente, obtenemos la ecuación numérica 31, para utilizarla en el software MATLAB:

$$P_{motor} = \frac{0.0274}{8.878e{-}12\ s^3 + 1.291e{-}05\ s^2 + 0.0007648\ s} \quad (31)$$

Para poder aplicar el RRA al problema es necesaria una función de Matlab con la que podamos evaluar la aptitud o el "coste" de las plantas. Esta función es la función *cost* representada en Simulink en la Figura 44.

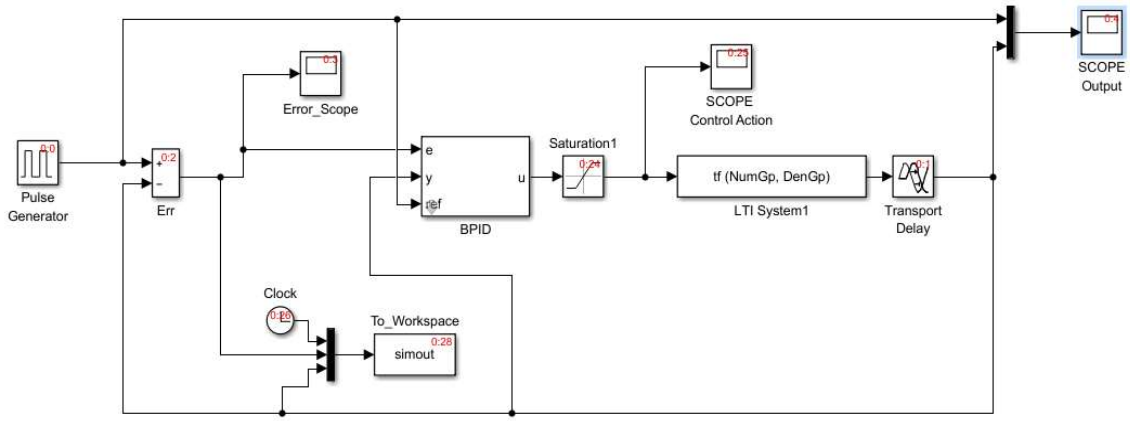


Figura 44. Modelo de Simulink de la función *cost*

El problema consiste en ajustar las ganancias del controlador BPID (Kb, Kp, Ki, Kd) tal que el error entre la entrada en escalón y la salida del sistema se minimice lo máximo posible. Por tanto, utilizaremos el RRA para que busque una combinación de estos con la cual se minimice la función coste entre una población inicial de valores.

Para ello, hay que tener en cuenta que el valor de Kb nunca puede ser superior a 1. Por lo tanto, este parámetro se limita en todo el trabajo con un valor máximo de 1.

Parámetros del algoritmo

El número de plantas que define la población del algoritmo será igual a 10. Cuanto mayor sea el tamaño de la población mayor será el número de combinaciones posibles de los parámetros a calcular.

Esta población se generará de manera aleatoria inicialmente. El rango entre el que están estos valores depende del valor máximo de cada parámetro que se establece antes de ejecutar el algoritmo. Los valores máximos que se van a utilizar para cada parámetro son los siguientes:

- Máximo valor de Kp = 1
- Máximo valor de Ki = 1
- Máximo valor de Kd = 1
- Máximo valor de Kb = 1

Los parámetros constantes que se quieren ajustar son aquellos que influyen en el cálculo de la probabilidad directa o indirectamente:

- d_{runner}
- d_{root}
- tol
- a

Analizando estos parámetros se puede ajustar la precisión del algoritmo y disminuir aún más el coste de la función, consiguiendo un mejor control del sistema reduciendo el error de salida.

El ajuste de los valores tol y a, se mantienen iguales a los de la configuración del apartado anterior, puesto que a pesar de influir en el funcionamiento del algoritmo y su ejecución, no influyen en los estados de búsqueda tanto global como local.

Tras realizar las primeras evaluaciones, se observa una gran oscilación de salida que limita bastante el funcionamiento del motor. Esto puede ser debido al valor de la parte derivativa del controlador. Es por ello que se reduce su valor para estas primeras iteraciones. En concreto el valor de Kd se ajusta como $Kd = 0.2$

En un primer momento, se evalúan los parámetros de longitud (d_{runner} y d_{root}), por lo que se efectúan las siguientes combinaciones entre ambos para poder escoger entre ellas cuál es la que mayor beneficio proporciona:

- d_{runner} = 0.3 y d_{root} = 0.01
- d_{runner} = 0.5 y d_{root} = 0.01
- d_{runner} = 0.7 y d_{root} =0.01

d_{runner}	Coste medio	Mejor coste
0.3	0.037699	0.035607
0.5	0.035264	0.031031
0.7	0.035593	0.033827

Tabla 2. Resultados obtenidos con diferentes valores del parámetro drunner

Como se puede observar en la Tabla 2, los tres valores resultan aptos para aplicarlos al problema, consiguiendo un mejor resultado con el valor 0,5. Por tanto, se puede fijar este valor para seguir operando con el parámetro d_{root}. Tras concretar cuál es la longitud de tallos (d_{runner}) más apropiada, se realiza la misma operación para la longitud de las raíces (d_{root}) con los resultados obtenidos en la Tabla 3, donde se muestra el valor 0,1 como mejor candidato:

- d_{root} = 0.05 y d_{runner} = 0.5
- d_{root} = 0.01 y d_{runner} = 0.5
- d_{root} = 0.003 y d_{runner} = 0.5

d_{root}	Coste medio	Mejor coste
0.05	0.034458	0.034290
0.01	0.035264	0.031031
0.003	0.035363	0.033908

Tabla 3. Resultados obtenidos con diferentes parámetros del valor droot

Evolución de las iteraciones

Para evaluar la influencia de los parámetros, también es importante fijarse en cómo evolucionan las iteraciones a lo largo de la ejecución del algoritmo. Para ello se van a realizar cinco pruebas con las diferentes combinaciones de parámetros anteriores:

La evolución de las iteraciones con d_{runner} = 0.7 y d_{root} = 0.01 se representa en la Figura 45.

```
It: 1.000000 Best 0.122236
Parameters: Kp: 0.724992 Ki: 0.607416 Kd: 0.117673 Kb: 0.000000

It: 2.000000 Best 0.089256
Parameters: Kp: 0.898935 Ki: 0.654905 Kd: 0.089528 Kb: 0.001000

It: 3.000000 Best 0.071084
Parameters: Kp: 0.773987 Ki: 0.651774 Kd: 0.047571 Kb: 0.358802

It: 4.000000 Best 0.043476
Parameters: Kp: 0.843639 Ki: 0.369353 Kd: 0.022892 Kb: 0.001000

It: 5.000000 Best 0.043476
Parameters: Kp: 0.843639 Ki: 0.369353 Kd: 0.022892 Kb: 0.001000

It: 6.000000 Best 0.043052
Parameters: Kp: 0.844727 Ki: 0.369353 Kd: 0.022892 Kb: 0.001000

It: 7.000000 Best 0.039007
Parameters: Kp: 1.000000 Ki: 0.369353 Kd: 0.022892 Kb: 0.001000

It: 8.000000 Best 0.033827
Parameters: Kp: 0.998536 Ki: 0.101921 Kd: 0.015053 Kb: 0.280258

It: 9.000000 Best 0.033827
Parameters: Kp: 0.998536 Ki: 0.101921 Kd: 0.015053 Kb: 0.280258

It: 10.000000 Best 0.033827
Parameters: Kp: 0.998536 Ki: 0.101921 Kd: 0.015053 Kb: 0.280258
```

```
It: 1.000000 Best 0.064523
Parameters: Kp: 0.604314 Ki: 0.516432 Kd: 0.001501 Kb: 0.000000

It: 2.000000 Best 0.060037
Parameters: Kp: 0.627592 Ki: 0.516937 Kd: 0.002421 Kb: 0.000960

It: 3.000000 Best 0.055962
Parameters: Kp: 0.925700 Ki: 0.685528 Kd: 0.035690 Kb: 0.313801

It: 4.000000 Best 0.038731
Parameters: Kp: 1.000000 Ki: 0.575168 Kd: 0.012537 Kb: 0.313801

It: 5.000000 Best 0.038731
Parameters: Kp: 1.000000 Ki: 0.575168 Kd: 0.012537 Kb: 0.313801

It: 6.000000 Best 0.038731
Parameters: Kp: 1.000000 Ki: 0.575168 Kd: 0.012537 Kb: 0.313801

It: 7.000000 Best 0.038731
Parameters: Kp: 1.000000 Ki: 0.575168 Kd: 0.012537 Kb: 0.313801

It: 8.000000 Best 0.037886
Parameters: Kp: 1.000000 Ki: 0.408721 Kd: 0.012537 Kb: 0.578631

It: 9.000000 Best 0.037886
Parameters: Kp: 1.000000 Ki: 0.408721 Kd: 0.012537 Kb: 0.578631

It: 10.000000 Best 0.037359
Parameters: Kp: 1.000000 Ki: 0.408721 Kd: 0.012537 Kb: 0.275305
```

Figura 45. Evolución de las iteraciones (I)

Con esta configuración de parámetros iniciales, se observa en ambas simulaciones que el algoritmo funciona de manera correcta, pues los costes se van disminuyendo a medida que avanzan las iteraciones.

Además, a pesar de obtener la misma solución durante varias iteraciones, termina encontrando una mejor. Esto puede deberse a la capacidad del algoritmo de escapar de mínimos locales, o a la capacidad de profundizar en la búsqueda local de recursos.

La evolución de las iteraciones con d_{runner} = 0.3 y d_{root} = 0.01 se representa en la Figura 46.

```
It: 1.000000 Best 0.071361
Parameters: Kp: 0.744868 Ki: 0.892267 Kd: 0.048521 Kb: 0.000000

It: 2.000000 Best 0.051343
Parameters: Kp: 0.778110 Ki: 0.892267 Kd: 0.024825 Kb: 0.096433

It: 3.000000 Best 0.051343
Parameters: Kp: 0.778110 Ki: 0.892267 Kd: 0.024825 Kb: 0.096433

It: 4.000000 Best 0.051343
Parameters: Kp: 0.778110 Ki: 0.892267 Kd: 0.024825 Kb: 0.096433

It: 5.000000 Best 0.047035
Parameters: Kp: 0.896540 Ki: 0.654224 Kd: 0.002714 Kb: 0.173207

It: 6.000000 Best 0.042935
Parameters: Kp: 0.896540 Ki: 0.651421 Kd: 0.014765 Kb: 0.177792

It: 7.000000 Best 0.042112
Parameters: Kp: 0.896540 Ki: 0.589712 Kd: 0.014765 Kb: 0.177792

It: 8.000000 Best 0.042112
Parameters: Kp: 0.896540 Ki: 0.589712 Kd: 0.014765 Kb: 0.177792

It: 9.000000 Best 0.040849
Parameters: Kp: 1.000000 Ki: 0.682410 Kd: 0.013859 Kb: 0.222263

It: 10.000000 Best 0.039791
Parameters: Kp: 1.000000 Ki: 0.541209 Kd: 0.013859 Kb: 0.081885
```

```
It: 1.000000 Best 0.058424
Parameters: Kp: 0.788891 Ki: 0.092398 Kd: 0.047574 Kb: 0.000000

It: 2.000000 Best 0.047570
Parameters: Kp: 0.713917 Ki: 0.220700 Kd: 0.021689 Kb: 0.001000

It: 3.000000 Best 0.046351
Parameters: Kp: 0.713917 Ki: 0.220700 Kd: 0.020900 Kb: 0.001000

It: 4.000000 Best 0.042320
Parameters: Kp: 0.860672 Ki: 0.217675 Kd: 0.020900 Kb: 0.001000

It: 5.000000 Best 0.038389
Parameters: Kp: 0.930834 Ki: 0.167272 Kd: 0.020900 Kb: 0.001000

It: 6.000000 Best 0.038389
Parameters: Kp: 0.930834 Ki: 0.167272 Kd: 0.020900 Kb: 0.001000

It: 7.000000 Best 0.036231
Parameters: Kp: 1.000000 Ki: 0.336429 Kd: 0.007254 Kb: 0.001000

It: 8.000000 Best 0.035805
Parameters: Kp: 1.000000 Ki: 0.452538 Kd: 0.007254 Kb: 0.001000

It: 9.000000 Best 0.035805
Parameters: Kp: 1.000000 Ki: 0.452538 Kd: 0.007254 Kb: 0.001000

It: 10.000000 Best 0.035607
Parameters: Kp: 1.000000 Ki: 0.371379 Kd: 0.007254 Kb: 0.001000
```

Figura 46. Evolución de las iteraciones (II)

En es este caso también se observa que el funcionamiento del algoritmo es correcto, obteniendo un coste muy similar al caso anterior.

La evolución de las iteraciones con d_{runner} = 0.5 y d_{root} = 0.01 se representa en la Figura 47.

```
It: 1.000000 Best 0.079253
Parameters: Kp: 0.715045 Ki: 0.856182 Kd: 0.056302 Kb: 0.000000

It: 2.000000 Best 0.055317
Parameters: Kp: 1.000000 Ki: 0.379862 Kd: 0.047603 Kb: 0.152952

It: 3.000000 Best 0.055038
Parameters: Kp: 1.000000 Ki: 0.376249 Kd: 0.047603 Kb: 0.152952

It: 4.000000 Best 0.047750
Parameters: Kp: 1.000000 Ki: 0.414628 Kd: 0.000189 Kb: 0.126217

It: 5.000000 Best 0.046507
Parameters: Kp: 1.000000 Ki: 0.162275 Kd: 0.038276 Kb: 0.236418

It: 6.000000 Best 0.045989
Parameters: Kp: 1.000000 Ki: 0.164441 Kd: 0.038276 Kb: 0.236418

It: 7.000000 Best 0.045873
Parameters: Kp: 1.000000 Ki: 0.163423 Kd: 0.038276 Kb: 0.236418

It: 8.000000 Best 0.041095
Parameters: Kp: 1.000000 Ki: 0.163423 Kd: 0.031276 Kb: 0.232321

It: 9.000000 Best 0.040867
Parameters: Kp: 1.000000 Ki: 0.086870 Kd: 0.031276 Kb: 0.232321

It: 10.000000 Best 0.039497
Parameters: Kp: 1.000000 Ki: 0.001716 Kd: 0.031276 Kb: 0.232321
```

```
It: 1.000000 Best 0.133619
Parameters: Kp: 0.516997 Ki: 0.143156 Kd: 0.111874 Kb: 0.000000

It: 2.000000 Best 0.115513
Parameters: Kp: 0.516997 Ki: 0.001000 Kd: 0.104052 Kb: 0.000000

It: 3.000000 Best 0.112034
Parameters: Kp: 0.516997 Ki: 0.001000 Kd: 0.097685 Kb: 0.000000

It: 4.000000 Best 0.090670
Parameters: Kp: 0.582138 Ki: 0.046797 Kd: 0.074911 Kb: 0.188422

It: 5.000000 Best 0.067125
Parameters: Kp: 0.608504 Ki: 0.001000 Kd: 0.049061 Kb: 0.059994

It: 6.000000 Best 0.049004
Parameters: Kp: 0.630035 Ki: 0.001000 Kd: 0.027803 Kb: 0.060824

It: 7.000000 Best 0.044988
Parameters: Kp: 0.723354 Ki: 0.001000 Kd: 0.027036 Kb: 0.060824

It: 8.000000 Best 0.043762
Parameters: Kp: 0.723354 Ki: 0.002423 Kd: 0.027036 Kb: 0.064195

It: 9.000000 Best 0.032576
Parameters: Kp: 0.967574 Ki: 0.001000 Kd: 0.009130 Kb: 0.069899

It: 10.000000 Best 0.031031
Parameters: Kp: 1.000000 Ki: 0.001000 Kd: 0.008512 Kb: 0.069899
```

Figura 47. Evolución de las iteraciones (III)

Esta configuración es la que obtiene un mejor resultado de los costes, y es por ello que el valor de d_{runner} en el resto del problema se mantiene constante en el resto del problema.

La evolución de las iteraciones con d_{runner} = 0.5 y d_{root} = 0.05 se representa en la Figura 48.

```
It: 1.000000 Best 0.129276
Parameters: Kp: 0.831302 Ki: 0.790235 Kd: 0.142542 Kb: 0.000000

It: 2.000000 Best 0.103691
Parameters: Kp: 0.175788 Ki: 0.001000 Kd: 0.019686 Kb: 0.001000

It: 3.000000 Best 0.100074
Parameters: Kp: 0.181557 Ki: 0.001000 Kd: 0.019686 Kb: 0.021980

It: 4.000000 Best 0.097360
Parameters: Kp: 0.197743 Ki: 0.001000 Kd: 0.019686 Kb: 0.021980

It: 5.000000 Best 0.092540
Parameters: Kp: 0.203277 Ki: 0.001000 Kd: 0.018520 Kb: 0.024255

It: 6.000000 Best 0.064640
Parameters: Kp: 0.443776 Ki: 0.001000 Kd: 0.001000 Kb: 0.001000

It: 7.000000 Best 0.053076
Parameters: Kp: 0.457188 Ki: 0.006777 Kd: 0.005189 Kb: 0.006127

It: 8.000000 Best 0.045400
Parameters: Kp: 0.668129 Ki: 0.181553 Kd: 0.013903 Kb: 0.001000

It: 9.000000 Best 0.038030
Parameters: Kp: 0.844293 Ki: 0.001000 Kd: 0.017761 Kb: 0.221166

It: 10.000000 Best 0.034627
Parameters: Kp: 0.939283 Ki: 0.001000 Kd: 0.015290 Kb: 0.221166
```

```
It: 1.000000 Best 0.058679
Parameters: Kp: 0.832708 Ki: 0.259401 Kd: 0.042604 Kb: 0.000000

It: 2.000000 Best 0.052246
Parameters: Kp: 0.763172 Ki: 0.331504 Kd: 0.001000 Kb: 0.001000

It: 3.000000 Best 0.043788
Parameters: Kp: 0.976601 Ki: 0.085652 Kd: 0.033463 Kb: 0.134668

It: 4.000000 Best 0.037942
Parameters: Kp: 0.778728 Ki: 0.070531 Kd: 0.005369 Kb: 0.346022

It: 5.000000 Best 0.037025
Parameters: Kp: 0.802616 Ki: 0.050211 Kd: 0.005369 Kb: 0.351161

It: 6.000000 Best 0.036457
Parameters: Kp: 0.802616 Ki: 0.056126 Kd: 0.005369 Kb: 0.351161

It: 7.000000 Best 0.034570
Parameters: Kp: 1.000000 Ki: 0.170897 Kd: 0.010813 Kb: 0.393709

It: 8.000000 Best 0.034290
Parameters: Kp: 1.000000 Ki: 0.170897 Kd: 0.011926 Kb: 0.393709

It: 9.000000 Best 0.034290
Parameters: Kp: 1.000000 Ki: 0.170897 Kd: 0.011926 Kb: 0.393709

It: 10.000000 Best 0.034290
Parameters: Kp: 1.000000 Ki: 0.170897 Kd: 0.011926 Kb: 0.393709
```

Figura 48. Evolución de las iteraciones (IV)

Aumentando el valor de d_{root} se observa cómo, una vez más, el algoritmo trabaja de manera eficiente, sin embargo los costes no han mejorado con respecto al anterior caso, sino que empeoran.

La evolución de las iteraciones con d_{runner} = 0.5 y d_{root} = 0.003 se representa en la Figura 49.

```
It: 1.000000 Best 0.064330
Parameters: Kp: 0.967550 Ki: 1.000000 Kd: 0.053450 Kb: 0.037466

It: 2.000000 Best 0.056896
Parameters: Kp: 0.967550 Ki: 1.000000 Kd: 0.041971 Kb: 0.001000

It: 3.000000 Best 0.050225
Parameters: Kp: 0.966550 Ki: 1.000000 Kd: 0.001187 Kb: 0.001000

It: 4.000000 Best 0.050225
Parameters: Kp: 0.966550 Ki: 1.000000 Kd: 0.001187 Kb: 0.001000

It: 5.000000 Best 0.046356
Parameters: Kp: 0.862242 Ki: 0.902333 Kd: 0.004165 Kb: 0.187597

It: 6.000000 Best 0.042257
Parameters: Kp: 1.000000 Ki: 1.000000 Kd: 0.012257 Kb: 0.253047

It: 7.000000 Best 0.041597
Parameters: Kp: 0.998537 Ki: 1.000000 Kd: 0.012257 Kb: 0.253047

It: 8.000000 Best 0.039414
Parameters: Kp: 0.974205 Ki: 0.572222 Kd: 0.011270 Kb: 0.207078

It: 9.000000 Best 0.038709
Parameters: Kp: 0.974205 Ki: 0.429735 Kd: 0.011270 Kb: 0.423608

It: 10.000000 Best 0.036818
Parameters: Kp: 0.974734 Ki: 0.429735 Kd: 0.011270 Kb: 0.423608
```

```
It: 1.000000 Best 0.073202
Parameters: Kp: 0.348235 Ki: 0.165471 Kd: 0.005627 Kb: 0.000000

It: 2.000000 Best 0.073202
Parameters: Kp: 0.348235 Ki: 0.165471 Kd: 0.005627 Kb: 0.000000

It: 3.000000 Best 0.066500
Parameters: Kp: 0.398405 Ki: 0.001000 Kd: 0.001000 Kb: 0.001000

It: 4.000000 Best 0.052144
Parameters: Kp: 0.569270 Ki: 0.001000 Kd: 0.001000 Kb: 0.001000

It: 5.000000 Best 0.042172
Parameters: Kp: 0.814568 Ki: 0.001000 Kd: 0.001058 Kb: 0.001000

It: 6.000000 Best 0.035121
Parameters: Kp: 0.915490 Ki: 0.001000 Kd: 0.012737 Kb: 0.001000

It: 7.000000 Best 0.034755
Parameters: Kp: 0.914258 Ki: 0.001000 Kd: 0.012737 Kb: 0.001000

It: 8.000000 Best 0.034755
Parameters: Kp: 0.914258 Ki: 0.001000 Kd: 0.012737 Kb: 0.001000

It: 9.000000 Best 0.034755
Parameters: Kp: 0.914258 Ki: 0.001000 Kd: 0.012737 Kb: 0.001000

It: 10.000000 Best 0.033908
Parameters: Kp: 0.914258 Ki: 0.001096 Kd: 0.012737 Kb: 0.001000
```

Figura 49. Evolución de las iteraciones (V)

Probando a reducir en este caso el valor de d_{root}, también se obtienen resultados similares aunque en ningún caso mejores a la configuración con d_{root} =0.01

Tras estas pruebas se puede concluir que los valores que mejor coste proporcionan son los que se corresponden a la configuración con d_{runner} = 0.5 y d_{root} = 0.01

Simulación del sistema introduciendo los parámetros del controlador BPID

El algoritmo ha sido implementado utilizando Matlab, y el modelo del motor de corriente continua utilizando la herramienta de Matlab Simulink. Para comprobar si el sistema está controlado adecuadamente se van a realizar varias pruebas analizando la salida del sistema respecto a la entrada cuando se introducen en el controlador BPID los valores obtenidos con el algoritmo.

Para d_{runner} = 0.5 y d_{root} = 0.01 se obtiene la Figura 50.

Parameters: Kp: 1.000000 Ki: 0.001000 Kd: 0.008512 Kb: 0.069899

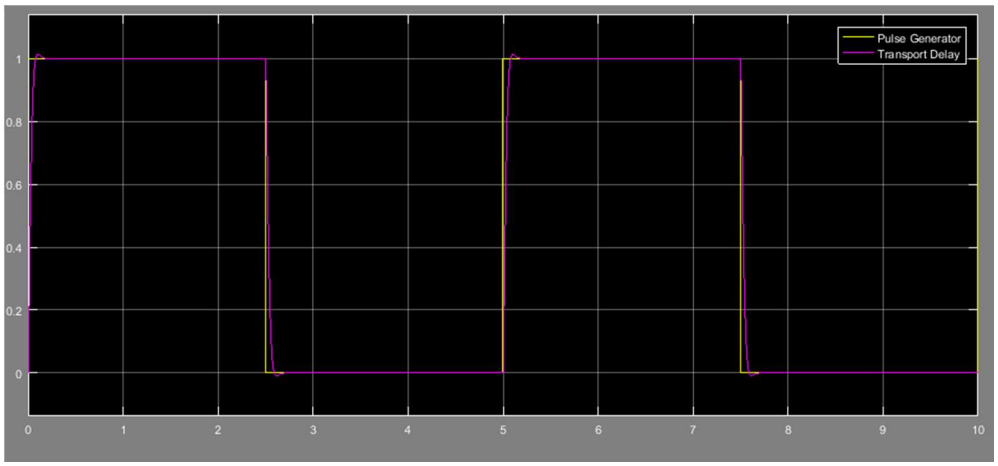


Figura 50. Simulación de la señal controlada (I)

Para d_{runner} = 0.3 y d_{root} = 0.01 se obtiene la Figura 51.

Parameters: Kp: 1.000000 Ki: 0.371379 Kd: 0.007254 Kb: 0.001000

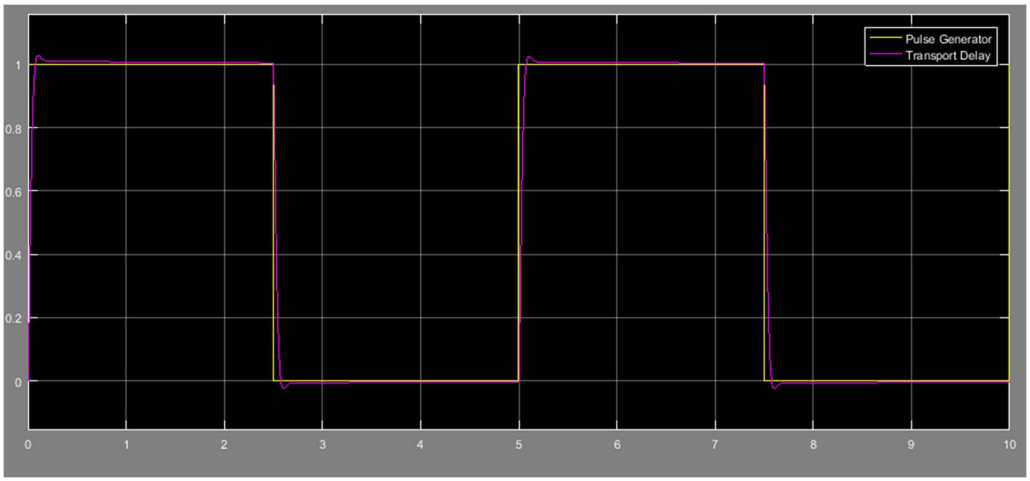


Figura 51. Simulación de la señal controlada (II)

Para d_{runner} = 0.7 y d_{root} = 0.01 se obtiene la Figura 52.

Parameters: Kp: 0.998536 Ki: 0.101921 Kd: 0.015053 Kb: 0.280258

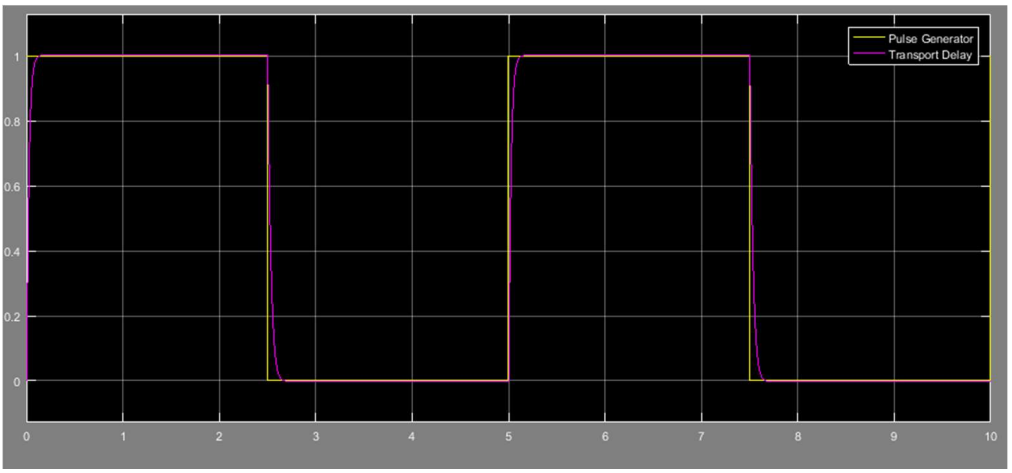


Figura 52. Simulación de la señal controlada (III)

Para d_{runner} = 0.5 y d_{root} = 0.05 se obtiene la Figura 53.

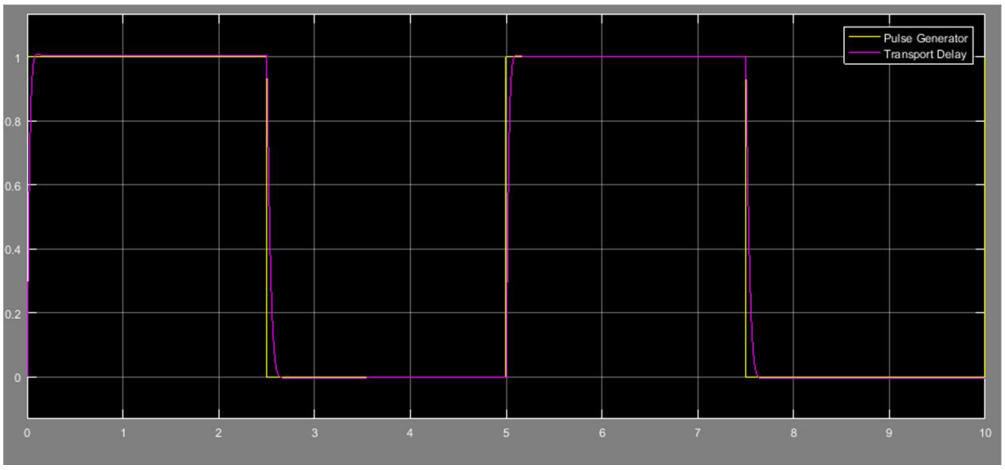


Figura 53. Simulación de la señal controlada (IV)

Para d_{runner} = 0.5 y d_{root} = 0.003 se obtiene la Figura 54.

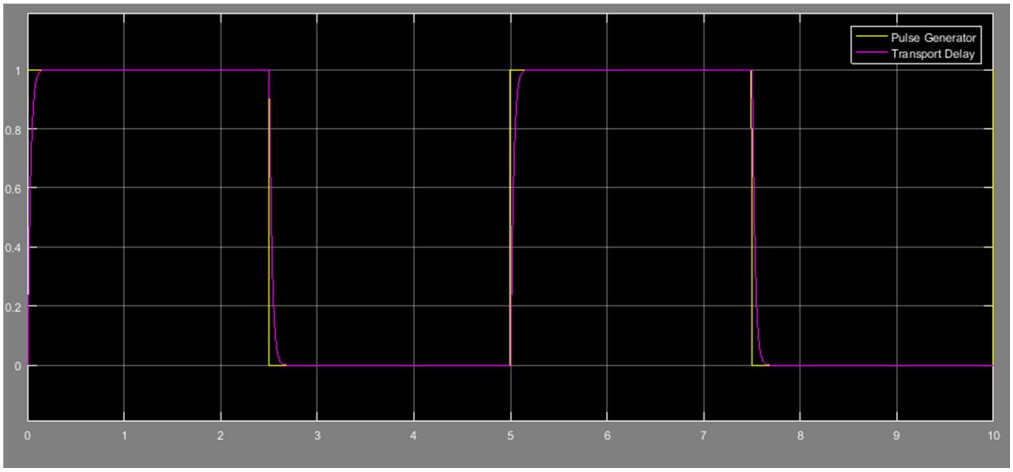


Figura 54. Simulación de la señal controlada (V)

Una vez que el algoritmo ha quedado ajustado, se procede a variar los límites superiores de las constantes de control. En la Tabla 4 se muestra cómo varía el coste cuando se incrementan los valores máximos de Kp, Ki y Kd.

Valor máximo (Kp, Ki, Kd, Kb)	Coste	Kp	Ki	Kd	Kb
(1, 1, 0.2, 1)	0.031031	1.000000	0.001000	0.008512	0.069899
(10, 10, 0.5, 1)	0.020964	8.292774	0.239535	0.045231	0.001000
(100, 100, 5, 1)	0.008116	77.471969	0.001000	0.310340	0.055577
(500, 500, 5, 1)	0.022212	6.763540	0.001000	0.031381	0.007570

Tabla 4. Resultados obtenidos con los distintos límites para las ganancias del controlador

Como referencia se vuelven a mostrar los resultados del estudio anterior para compararlos con los nuevos resultados.

Con los valores máximos de Kp, Ki, Kb = 1; Kd = 0.2 se obtiene la Figura 55.

```
It: 1.000000 Best 0.133619
Parameters: Kp: 0.516997 Ki: 0.143156 Kd: 0.111874 Kb: 0.000000

It: 2.000000 Best 0.115513
Parameters: Kp: 0.516997 Ki: 0.001000 Kd: 0.104052 Kb: 0.000000

It: 3.000000 Best 0.112034
Parameters: Kp: 0.516997 Ki: 0.001000 Kd: 0.097685 Kb: 0.000000

It: 4.000000 Best 0.090670
Parameters: Kp: 0.582138 Ki: 0.046797 Kd: 0.074911 Kb: 0.188422

It: 5.000000 Best 0.067125
Parameters: Kp: 0.608504 Ki: 0.001000 Kd: 0.049061 Kb: 0.059994

It: 6.000000 Best 0.049004
Parameters: Kp: 0.630035 Ki: 0.001000 Kd: 0.027803 Kb: 0.060824

It: 7.000000 Best 0.044988
Parameters: Kp: 0.723354 Ki: 0.001000 Kd: 0.027036 Kb: 0.060824

It: 8.000000 Best 0.043762
Parameters: Kp: 0.723354 Ki: 0.002423 Kd: 0.027036 Kb: 0.064195

It: 9.000000 Best 0.032576
Parameters: Kp: 0.967574 Ki: 0.001000 Kd: 0.009130 Kb: 0.069899

It: 10.000000 Best 0.031031
Parameters: Kp: 1.000000 Ki: 0.001000 Kd: 0.008512 Kb: 0.069899
```

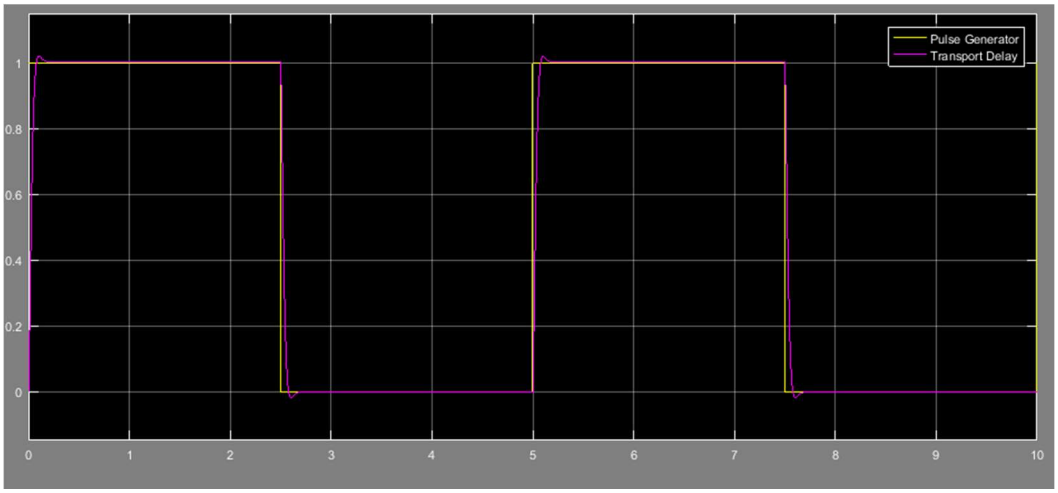


Figura 55. Evolución de las iteraciones y simulación de la señal controlada (I)

Aumentando el valor máximo de Kp y Ki a 10, se puede también aumentar el límite superior de Kd en 0.5, tratando de evitar las sobreocilaciones.

Con los valores máximos de Kp, Ki = 10; Kd = 0.5; Kb = 1 se obtiene la Figura 56.

```
It: 1.000000 Best 0.024015
Parameters: Kp: 6.030747 Ki: 1.840149 Kd: 0.043729 Kb: 0.000000

It: 2.000000 Best 0.021835
Parameters: Kp: 8.326508 Ki: 1.840149 Kd: 0.045231 Kb: 0.000937

It: 3.000000 Best 0.021835
Parameters: Kp: 8.326508 Ki: 1.840149 Kd: 0.045231 Kb: 0.000937

It: 4.000000 Best 0.020964
Parameters: Kp: 8.292774 Ki: 0.239535 Kd: 0.045231 Kb: 0.001000

It: 5.000000 Best 0.020964
Parameters: Kp: 8.292774 Ki: 0.239535 Kd: 0.045231 Kb: 0.001000

It: 6.000000 Best 0.020964
Parameters: Kp: 8.292774 Ki: 0.239535 Kd: 0.045231 Kb: 0.001000

It: 7.000000 Best 0.020964
Parameters: Kp: 8.292774 Ki: 0.239535 Kd: 0.045231 Kb: 0.001000

It: 8.000000 Best 0.020964
Parameters: Kp: 8.292774 Ki: 0.239535 Kd: 0.045231 Kb: 0.001000

It: 9.000000 Best 0.020964
Parameters: Kp: 8.292774 Ki: 0.239535 Kd: 0.045231 Kb: 0.001000

It: 10.000000 Best 0.020964
Parameters: Kp: 8.292774 Ki: 0.239535 Kd: 0.045231 Kb: 0.001000
```

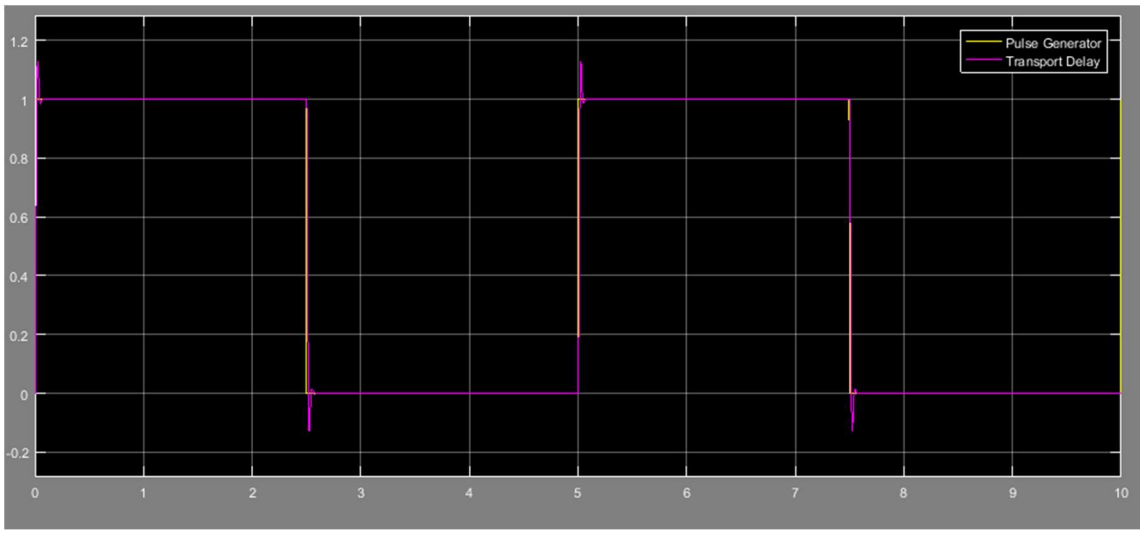


Figura 56. Evolución de las iteraciones y simulación de la señal controlada (II)

Como se puede observar en este caso, el error se reduce bastante y el sistema responde de manera más rápida. Sin embargo, se muestra una pequeña sobreoscilación en el proceso.

Aumentando nuevamente los valores máximos de Kp y Ki en 100 se pueden obtener mejores resultados todavía. A su vez, el valor máximo de Kd aumenta tan solo a 5.

Con los valores máximos de Kp, Ki = 100; Kd = 5; Kb = 1 se obtiene la Figura 57.

```
It: 1.000000 Best 0.024624
Parameters: Kp: 70.980312 Ki: 46.486447 Kd: 0.566356 Kb: 0.00000

It: 2.000000 Best 0.009997
Parameters: Kp: 79.616176 Ki: 40.103061 Kd: 0.330595 Kb: 0.00100

It: 3.000000 Best 0.009621
Parameters: Kp: 78.232789 Ki: 39.775817 Kd: 0.330595 Kb: 0.00100

It: 4.000000 Best 0.009248
Parameters: Kp: 78.038613 Ki: 23.657277 Kd: 0.330595 Kb: 0.00100

It: 5.000000 Best 0.008638
Parameters: Kp: 77.753697 Ki: 10.108425 Kd: 0.330595 Kb: 0.00669

It: 6.000000 Best 0.008250
Parameters: Kp: 77.753697 Ki: 0.001000 Kd: 0.310340 Kb: 0.050981

It: 7.000000 Best 0.008224
Parameters: Kp: 77.753697 Ki: 0.001000 Kd: 0.310340 Kb: 0.055577

It: 8.000000 Best 0.008224
Parameters: Kp: 77.753697 Ki: 0.001000 Kd: 0.310340 Kb: 0.055577

It: 9.000000 Best 0.008224
Parameters: Kp: 77.753697 Ki: 0.001000 Kd: 0.310340 Kb: 0.055577

It: 10.000000 Best 0.008116
Parameters: Kp: 77.471969 Ki: 0.001000 Kd: 0.310340 Kb: 0.055577
```

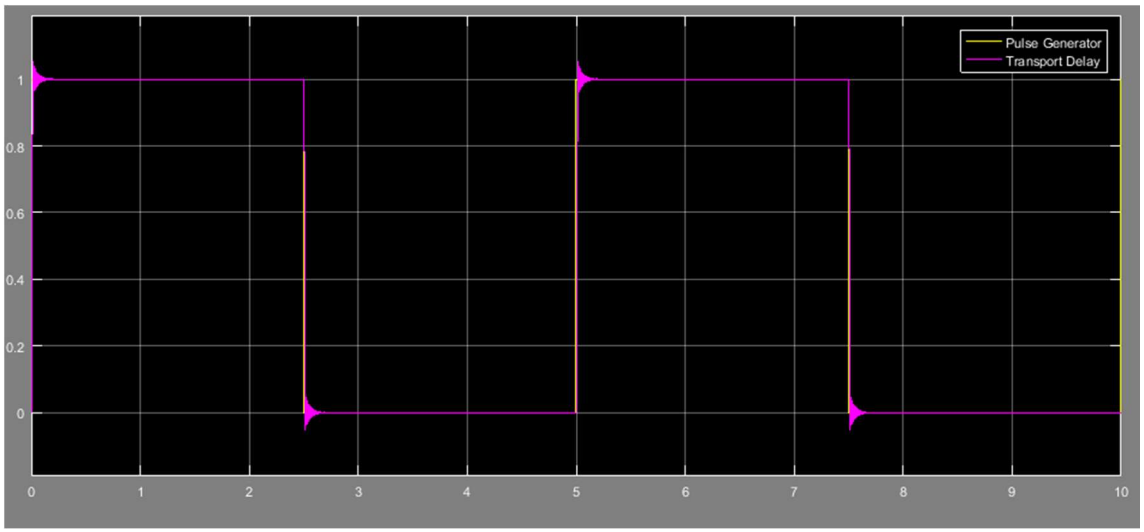


Figura 57. Evolución de las iteraciones y simulación de la señal controlada (III)

Con esta configuración se consigue un coste bastante más favorable que en los casos anteriores. Como se puede observar en la simulación, el controlador responde de manera eficiente y rápida, a pesar de mostrar una cierta sobreoscilación, la cual se ve compensada inmediatamente.

Como última prueba, se aumentan los valores máximos de Kp y Ki hasta 500, manteniendo el límite de Kd en 5.

Con los valores máximos de Kp, Ki = 500; Kd = 5; Kb = 1 se obtiene la Figura 58.

```
It: 1.000000 Best 0.028944
Parameters: Kp: 117.528418 Ki: 137.730538 Kd: 0.951607 Kb: 0.00000

It: 2.000000 Best 0.028025
Parameters: Kp: 115.695690 Ki: 154.890532 Kd: 0.739665 Kb: 0.00000

It: 3.000000 Best 0.027013
Parameters: Kp: 6.632636 Ki: 159.362499 Kd: 0.031381 Kb: 0.004761

It: 4.000000 Best 0.025488
Parameters: Kp: 6.632636 Ki: 49.150825 Kd: 0.031381 Kb: 0.004761

It: 5.000000 Best 0.024911
Parameters: Kp: 6.763540 Ki: 49.150825 Kd: 0.031381 Kb: 0.007570

It: 6.000000 Best 0.024911
Parameters: Kp: 6.763540 Ki: 49.150825 Kd: 0.031381 Kb: 0.007570

It: 7.000000 Best 0.022212
Parameters: Kp: 6.763540 Ki: 0.001000 Kd: 0.031381 Kb: 0.007570

It: 8.000000 Best 0.022212
Parameters: Kp: 6.763540 Ki: 0.001000 Kd: 0.031381 Kb: 0.007570

It: 9.000000 Best 0.022212
Parameters: Kp: 6.763540 Ki: 0.001000 Kd: 0.031381 Kb: 0.007570

It: 10.000000 Best 0.022212
Parameters: Kp: 6.763540 Ki: 0.001000 Kd: 0.031381 Kb: 0.007570
```

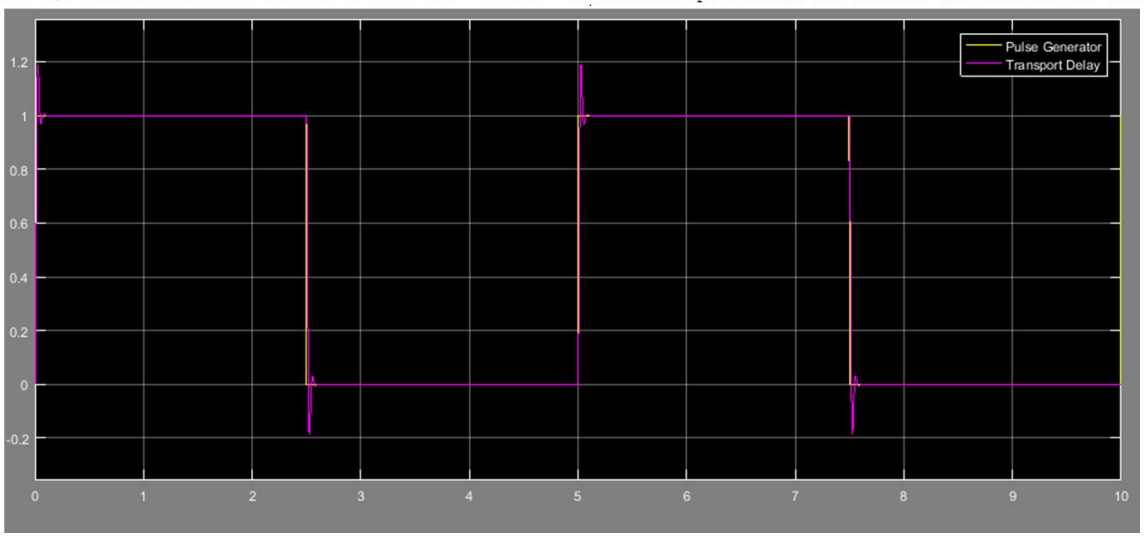


Figura 58. Evolución de las iteraciones y simulación de la señal controlada (IV)

En este caso, en lugar de seguir mejorando los costes, éstos se ven bastante empeorados. La sobreoscilación del sistema es demasiado grande. Además, al observar la evolución de las iteraciones, se muestra un gran salto en los valores de Kp y de Ki, lo que significa que unos valores muy altos no resultan aptos para el sistema.

Para este problema de control de la posición del motor de corriente continua, el algoritmo ofrece muy buenos resultados, pues consigue minimizar el coste a valores muy reducidos ajustando correctamente las ganancias del controlador BPID. Tras el estudio de los parámetros d_{runner} y d_{root} para un mejor ajuste, se puede concluir que mientras el valor de d_{runner} sea comparable a la diferencia entre los límites superior e inferior, el algoritmo funciona de manera aceptable en los procesos de búsqueda global y de búsqueda local por amplios pasos.

Así pues, el parámetro d_{root} también se ajusta de manera que sea un valor bastante más pequeño para profundizar en el proceso de búsqueda local por pequeños pasos. Mientras estos requisitos se cumplan el algoritmo consigue minimizar los costes. Sin embargo, para ajustar aún más los costes se seleccionan los valores que mejor resultados proporcionan.

Una vez se han seleccionado estos parámetros, se aprecia que los valores que influyen en los costes son los límites superiores para las ganancias del controlador, pues cuanto mayor sean estos, mejores costes se terminan obteniendo. A pesar de ello, hay que aumentar estos límites con un cierto control para que la salida del controlador sea correcta. Tras los resultados, se puede observar que la ganancia que más influye en estos, es la ganancia proporcional, pues es la que más se puede aumentar sin resultar perjudicial para el resultado, como puede ocurrir con el resto de ganancias.

Al finalizar los resultados se observa un límite en el valor máximo de las ganancias en el que los costes aumentan en lugar de reducirse. Esto se debe a que el controlador no logra compensar la sobreoscilación de manera adecuada.

Además, la salida alcanza la señal de referencia y se estabiliza prácticamente de manera automática pues como se ve en las gráficas, apenas hay diferencias entre ambas.

5.3 Control de velocidad de un motor de corriente continua

Modelo del motor

En esta sección se va a considerar el problema de control de la velocidad de un motor de corriente continua [20], y se va a utilizar el algoritmo RRA para encontrar una combinación de parámetros de un controlador BPID que minimice la función coste.

Las ecuaciones del motor son las mismas que en el apartado anterior, sin embargo sus propiedades no lo son, y su función de transferencia continua en lazo abierto teniendo en cuenta el voltaje como la entrada y la velocidad angular como la salida es la ecuación 32.

(32) $$\mathrm{P(s)} = \frac{\dot{\Theta}(s)}{\mathrm{V}(s)} = \frac{\mathrm{K}}{(\mathrm{Js}+\ )(\mathrm{Ls}+\mathrm{R})+\mathrm{K}^2}$$

Los parámetros físicos para este ejemplo son:

(J)	moment of inertia of the rotor	$0.01\ kg \cdot m^2$
(b)	motor viscous friction constant	$0.1\ N \cdot m \cdot s$
(Ke)	electromotive force constant	$0.01\ \frac{V}{rad/s}$
(Kt)	motor torque constant	$0.01\ \frac{N \cdot m}{Amp}$
(R)	electric resistance	$1\ Ohm$
(L)	electric inductance	$0.5\ H$

Se obtiene la función de transferencia de la ecuación 33:

(33) $$\mathrm{P_{motor}} = \frac{0.01}{0.005\ ^2+0.06\mathrm{s}+0.1001}$$

Control de velocidad del motor

Para comenzar con la simulación se utilizan los parámetros del algoritmo RRA con los que previamente se han obtenido mejores resultados. El valor máximo de las ganancias del controlador empieza siendo bajo, y poco a poco se aumenta si con ello es posible reducir el coste y conseguir unos parámetros más óptimos.

Durante todas las simulaciones se utiliza un tamaño de población de Np=5 y un número iteraciones stall_max = 10, con los parámetros del algoritmo:

- drunner = factores de 0.5
- droot = factores de 0.01
- tol = 0.01
- a = 0.1

De la misma manera que en el apartado anterior, se realiza un estudio de la evolución de las iteraciones para comprobar el comportamiento correcto del algoritmo, y se simula su señal para evaluar el funcionamiento del controlador. Se realizan varios estudios del sistema con distintos valores máximos de las constantes de control.

Para los valores máximos de Kp, Ki, Kd, Kb = 1 se obtiene la Figura 59.

Best: 0.450888

Parameters: Kp: 0.996147 Ki: 0.001000 Kd: 0.759177 Kb: 0.789684

```
It: 1.000000 Best 0.509285
Parameters: Kp: 0.880161 Ki: 0.503108 Kd: 0.559453 Kb: 0.247662

It: 2.000000 Best 0.503766
Parameters: Kp: 1.000000 Ki: 0.503108 Kd: 0.559453 Kb: 0.247662

It: 3.000000 Best 0.503548
Parameters: Kp: 0.998135 Ki: 0.503108 Kd: 0.559453 Kb: 0.247662

It: 4.000000 Best 0.495610
Parameters: Kp: 1.000000 Ki: 0.452887 Kd: 0.637110 Kb: 0.281084

It: 5.000000 Best 0.481144
Parameters: Kp: 1.000000 Ki: 0.299778 Kd: 0.611104 Kb: 0.281084

It: 6.000000 Best 0.476217
Parameters: Kp: 1.000000 Ki: 0.272945 Kd: 0.611104 Kb: 0.434266

It: 7.000000 Best 0.466016
Parameters: Kp: 1.000000 Ki: 0.191351 Kd: 0.527294 Kb: 0.678410

It: 8.000000 Best 0.454816
Parameters: Kp: 1.000000 Ki: 0.026184 Kd: 0.753927 Kb: 0.789684

It: 9.000000 Best 0.451255
Parameters: Kp: 1.000000 Ki: 0.001000 Kd: 0.761988 Kb: 0.789684

It: 10.000000 Best 0.450888
Parameters: Kp: 0.996147 Ki: 0.001000 Kd: 0.759177 Kb: 0.789684
```

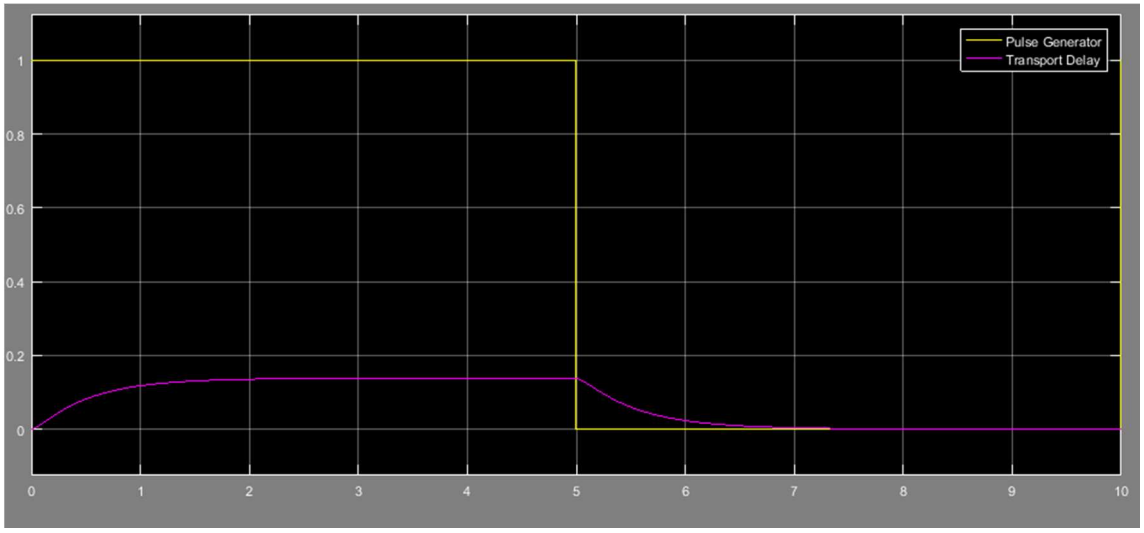


Figura 59. Evolución de las iteraciones y simulación de la señal controlada (V)

En este primer caso, se observa que el algoritmo está trabajando de manera correcta pues a lo largo de las iteraciones se van disminuyendo los costes. Sin embargo, el controlador no es el más correcto puesto que la señal de salida no logra alcanzar a la señal de referencia deseada. Es por esto por lo que se aumentan los valores máximos de Kp, Ki y Kd.

Con los valores máximos de Kp, Ki, Kd = 100; Kb = 1 se obtiene la Figura 60.

Best: 0.124872

Parameters: Kp: 100.000000 Ki: 13.183026 Kd: 14.700267 Kb: 0.349864

```
It: 1.000000 Best 0.158566
Parameters: Kp: 90.548733 Ki: 37.354117 Kd: 7.881650 Kb: 0.000000

It: 2.000000 Best 0.147633
Parameters: Kp: 90.548733 Ki: 28.880295 Kd: 14.700267 Kb: 0.179013

It: 3.000000 Best 0.126651
Parameters: Kp: 90.548733 Ki: 13.183026 Kd: 14.700267 Kb: 0.180784

It: 4.000000 Best 0.126651
Parameters: Kp: 90.548733 Ki: 13.183026 Kd: 14.700267 Kb: 0.180784

It: 5.000000 Best 0.125493
Parameters: Kp: 96.872457 Ki: 13.183026 Kd: 14.700267 Kb: 0.349864

It: 6.000000 Best 0.125493
Parameters: Kp: 96.872457 Ki: 13.183026 Kd: 14.700267 Kb: 0.349864

It: 7.000000 Best 0.125493
Parameters: Kp: 96.872457 Ki: 13.183026 Kd: 14.700267 Kb: 0.349864

It: 8.000000 Best 0.125481
Parameters: Kp: 97.147069 Ki: 13.183026 Kd: 14.700267 Kb: 0.349864

It: 9.000000 Best 0.125407
Parameters: Kp: 97.318581 Ki: 13.183026 Kd: 14.700267 Kb: 0.349864

It: 10.000000 Best 0.124872
Parameters: Kp: 100.000000 Ki: 13.183026 Kd: 14.700267 Kb: 0.349864
```

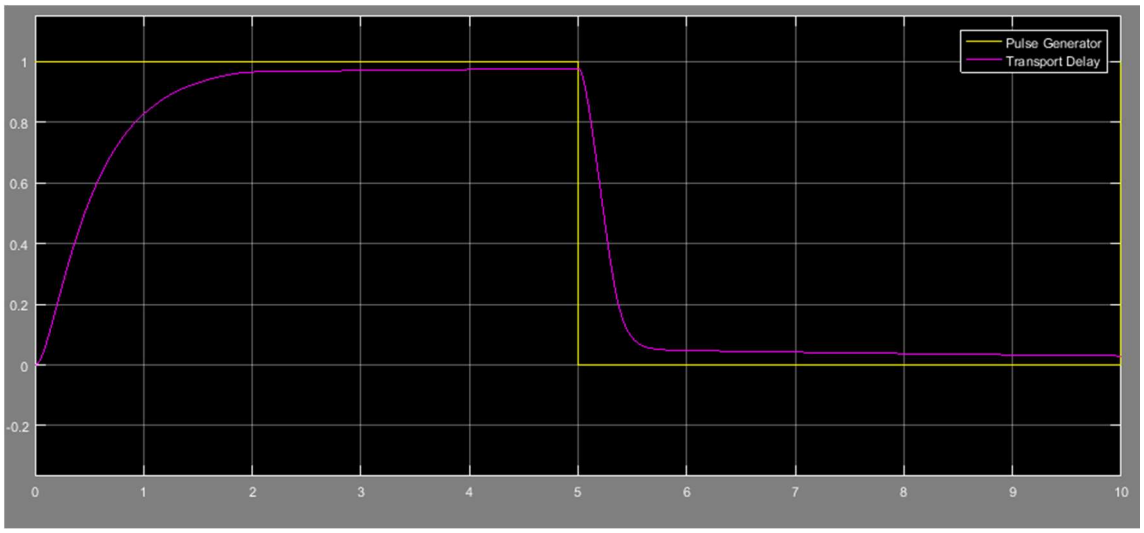


Figura 60. Evolución de las iteraciones y simulación de la señal controlada (VI)

Con estos nuevos parámetros se consigue un control del sistema más aceptable que en el caso anterior. La señal de salida se acerca mucho más a la de referencia y consigue estabilizarse. Aun así, el coste del sistema puede verse reducido aumentando todavía más los parámetros de control.

Valor máximo de Kp, Ki, Kd = 500; Kb = 1

Best: 0.094232

Parameters: Kp: 500.000000 Ki: 0.001000 Kd: 30.786332 Kb: 0.483686

```
It: 1.000000 Best 0.167293
Parameters: Kp: 344.700901 Ki: 265.437941 Kd: 94.426851 Kb: 0.000000

It: 2.000000 Best 0.160869
Parameters: Kp: 344.700901 Ki: 194.593762 Kd: 125.211696 Kb: 0.002194

It: 3.000000 Best 0.156007
Parameters: Kp: 360.402136 Ki: 312.925608 Kd: 53.936927 Kb: 0.258241

It: 4.000000 Best 0.153675
Parameters: Kp: 360.402136 Ki: 312.993396 Kd: 53.936927 Kb: 0.163683

It: 5.000000 Best 0.112486
Parameters: Kp: 500.000000 Ki: 26.871638 Kd: 80.135453 Kb: 0.486729

It: 6.000000 Best 0.097376
Parameters: Kp: 500.000000 Ki: 0.001000 Kd: 80.135453 Kb: 0.486729

It: 7.000000 Best 0.097376
Parameters: Kp: 500.000000 Ki: 0.001000 Kd: 80.135453 Kb: 0.486729

It: 8.000000 Best 0.097376
Parameters: Kp: 500.000000 Ki: 0.001000 Kd: 80.135453 Kb: 0.486729

It: 9.000000 Best 0.096473
Parameters: Kp: 500.000000 Ki: 0.477210 Kd: 19.613462 Kb: 0.486729

It: 10.000000 Best 0.094232
Parameters: Kp: 500.000000 Ki: 0.001000 Kd: 30.786332 Kb: 0.483686
```

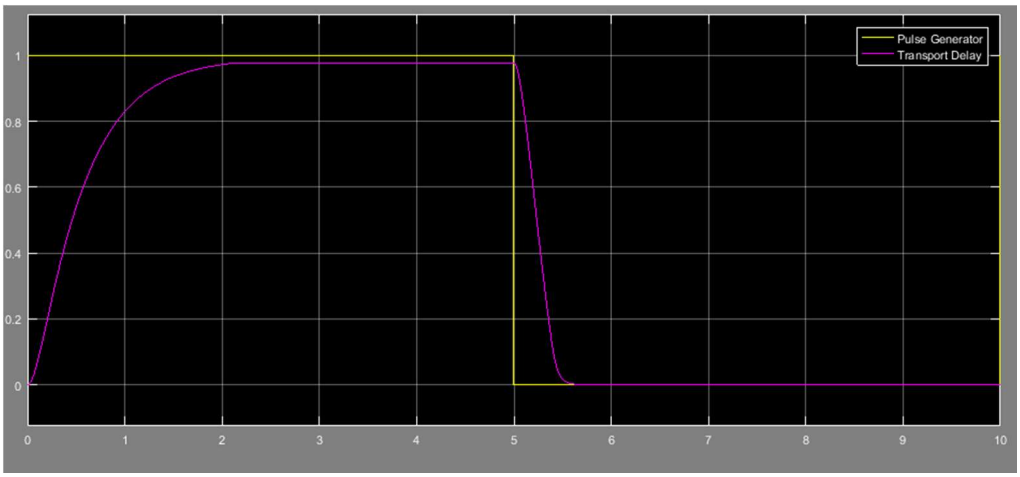


Figura 61. Evolución de las iteraciones y simulación de la señal controlada (VII)

Se puede observar que a medida que los valores de las constantes de control aumentan, el error disminuye y se consigue una señal de salida más estable, más rápida y más semejante a la señal de referencia que se pretende conseguir. En este caso, se puede apreciar que el valor que termina tomando Kp está en el límite, por lo que se podría aumentar aún más su valor para obtener mejores resultados.

Con los valores máximos de Kp = 1000; Ki,Kd=100 Kb=1 se obtiene la Figura 62.

Best: 0.089547

Parameters: Kp: 653.584953 Ki: 0.001000 Kd: 66.803082 Kb: 0.122581

```
It: 1.000000 Best 0.110178                                  It: 6.000000 Best 0.090564
Parameters: Kp: 944.106929 Ki: 51.664302 Kd: 44.529198 Kb: 0.000000 Parameters: Kp: 668.258970 Ki: 0.001000 Kd: 51.769443 Kb: 0.362137

It: 2.000000 Best 0.110178                                  It: 7.000000 Best 0.090019
Parameters: Kp: 944.106929 Ki: 51.664302 Kd: 44.529198 Kb: 0.000000 Parameters: Kp: 649.112999 Ki: 0.001000 Kd: 66.803082 Kb: 0.001000

It: 3.000000 Best 0.107938                                  It: 8.000000 Best 0.089863
Parameters: Kp: 710.140053 Ki: 34.728726 Kd: 45.737758 Kb: 0.166624 Parameters: Kp: 649.112999 Ki: 0.001000 Kd: 66.803082 Kb: 0.122581

It: 4.000000 Best 0.104748                                  It: 9.000000 Best 0.089547
Parameters: Kp: 710.140053 Ki: 18.960969 Kd: 53.286732 Kb: 0.344901 Parameters: Kp: 653.584953 Ki: 0.001000 Kd: 66.803082 Kb: 0.122581

It: 5.000000 Best 0.091536                                  It: 10.000000 Best 0.089547
Parameters: Kp: 718.873746 Ki: 0.001000 Kd: 49.252023 Kb: 0.155943 Parameters: Kp: 653.584953 Ki: 0.001000 Kd: 66.803082 Kb: 0.122581
```

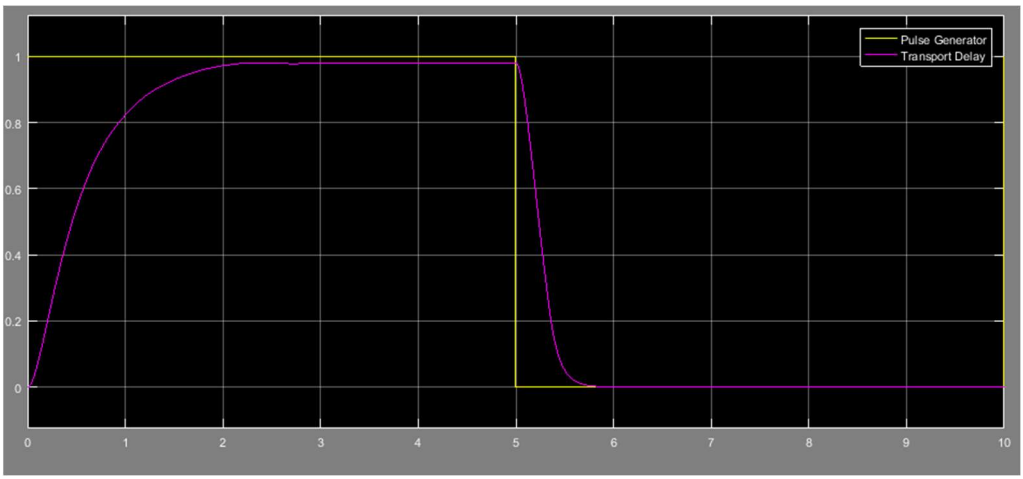


Figura 62. Evolución de las iteraciones y simulación de la señal controlada (VIII)

En este caso, es necesario limitar los valores de Kd y de Ki para evitar oscilaciones no deseadas para el control del sistema. Tras realizar la simulación de manera correcta, se obtienen unos resultados más favorables aún que en la anterior configuración.

Como última prueba se pueden volver a aumentar los parámetros de control para afinar completamente el sistema, a pesar de que los resultados ya obtenidos son unos valores propicios para el buen funcionamiento del sistema.

Con los valores máximos de Kp = 10000; Ki,Kd = 500 Kb = 1 se obtiene la Figura 63.

Best: 0.072578

Parameters: Kp: 6432.135397 Ki: 0.001000 Kd: 500.000000 Kb: 0.291906

```
It: 1.000000 Best 0.095399
Parameters: Kp: 3392.076645 Ki: 0.001000 Kd: 224.983338 Kb: 0.001000

It: 2.000000 Best 0.090345
Parameters: Kp: 5296.424160 Ki: 0.001000 Kd: 334.906745 Kb: 0.001000

It: 3.000000 Best 0.084533
Parameters: Kp: 3397.369016 Ki: 0.001000 Kd: 419.465739 Kb: 0.201676

It: 4.000000 Best 0.083063
Parameters: Kp: 4849.260722 Ki: 0.001000 Kd: 427.569015 Kb: 0.237150

It: 5.000000 Best 0.076271
Parameters: Kp: 5064.197484 Ki: 10.041518 Kd: 499.140024 Kb: 0.287855

It: 6.000000 Best 0.075581
Parameters: Kp: 6214.570744 Ki: 10.041518 Kd: 500.000000 Kb: 0.287855

It: 7.000000 Best 0.075518
Parameters: Kp: 6463.052456 Ki: 23.186925 Kd: 500.000000 Kb: 0.336179

It: 8.000000 Best 0.072616
Parameters: Kp: 6432.135397 Ki: 0.001000 Kd: 500.000000 Kb: 0.482315

It: 9.000000 Best 0.072578
Parameters: Kp: 6432.135397 Ki: 0.001000 Kd: 500.000000 Kb: 0.291906

It: 10.000000 Best 0.072578
Parameters: Kp: 6432.135397 Ki: 0.001000 Kd: 500.000000 Kb: 0.291906
```

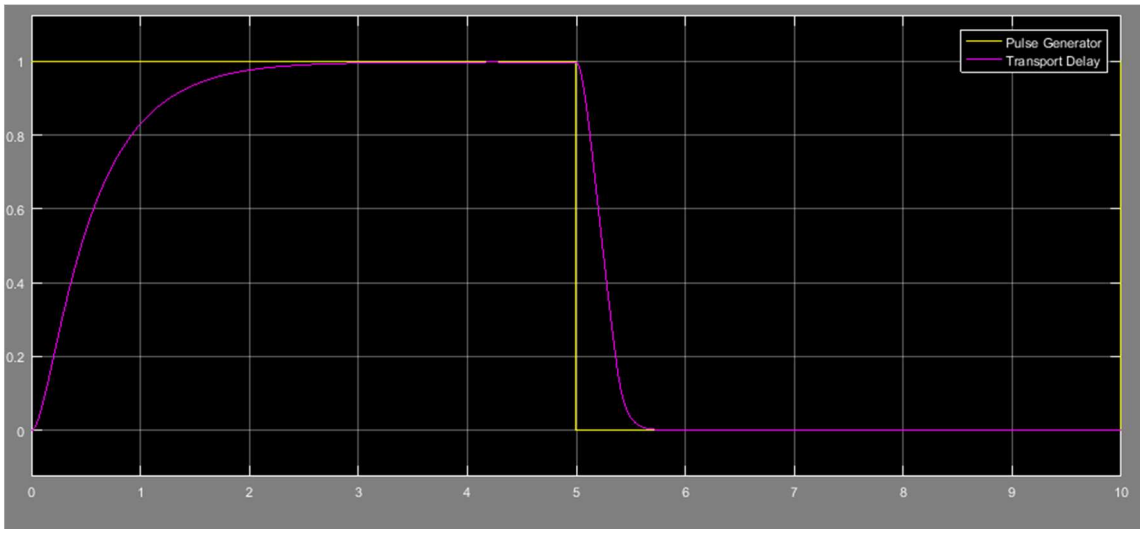


Figura 63. Evolución de las iteraciones y simulación de la señal controlada (IX)

Se puede ver en la anterior Figura que el sistema se ajusta con mayor precisión, residiendo el funcionamiento de control principalmente en las ganancias proporcional y derivativa.

Para el problema de control de la velocidad de un motor de corriente continua se consiguen obtener muy buenos resultados una vez más. Para este problema, se configuró el algoritmo RRA con los mismos valores de inicialización de sus parámetros que en el ejercicio anterior. Con esta configuración se observa que

el algoritmo también trabaja de manera correcta pero es necesario ajustar los límites de las ganancias. A medida que estos límites aumentan se obtienen mejores resultados, con señales de salida cada vez más parecidas a la señal de referencia.

En todas las pruebas, los resultados muestran valores muy cercanos al límite para la ganancia proporcional por lo que se puede aumentar su valor. También se pueden apreciar valores elevados en la ganancia derivativa, lo que puede suponer que el sistema sea controlable con un regulador PD, pues la parte integral se muestra bastante reducida.

La señal de salida en un primer momento está muy alejada de la señal de referencia, pero tras ajustar los límites de las ganancias se consigue alcanzar esta señal de manera estable y con un tiempo de estabilización en torno a los 2 segundos sin producir sobreoscilaciones, obteniendo unos buenos costes.

Capítulo 6. Conclusiones y aplicaciones futuras

El algoritmo RRA, como algoritmo genético basado en las técnicas de optimización metaheurísticas, es capaz de obtener buenos resultados una vez aplicado a las técnicas de control.

El algoritmo se fundamenta de la teoría de los algoritmos genéticos en la inicialización de una población que va sufriendo cambios aleatorios generación tras generación, como el paso de mutación o cruce de los algoritmos genéticos, y se reproduce de manera que sólo lo hacen aquellas soluciones con mejores resultados.

De este modo el algoritmo ha sido capaz de realizar tareas de optimización para la ingeniería de control al optimizar los parámetros de un controlador BPID en distintos ejemplos.

Es necesario configurar el algoritmo de manera que se ajuste a los parámetros del sistema, pero como se ha podido observar en los resultados experimentales del proyecto, tras realizar la configuración más apropiada, el algoritmo es capaz de obtener los mejores resultados posibles.

Para realizar esta tarea, el algoritmo toma como si fueran los cromosomas de la población los parámetros que se desean optimizar en la técnica de control. En concreto, cada cromosoma se corresponde con una de las ganancias de un controlador bilineal BPID.

Estos cromosomas, sin embargo, están limitados de manera inferior y superior mediante los parámetros Xu y Xl del algoritmo, los cuales influyen en gran medida en los resultados finales del problema. Con un amplio dominio del problema, las ganancias del controlador pueden tomar un mayor rango de valores para su aplicación. Sin embargo, es posible que en ciertos casos sea necesario limitar a propósito estos valores para que el sistema cumpla con las expectativas de control.

Por otro lado, el proceso de mutación del algoritmo viene dado por los valores d_{runner} y d_{root}, los cuales también son necesarios de ajustar para mejorar la eficiencia del funcionamiento del algoritmo. Estos valores tienen que estar en consonancia con los anteriores límites del dominio del problema, puesto que tienen que ser comparables a la diferencia entre el límite superior y el límite inferior para que la mutación sea eficaz, abarcando entre las posibles soluciones todo el dominio del problema, ofreciendo al algoritmo la posibilidad de escapar de un mínimo local y de profundizar en el mínimo global.

Este proyecto ha demostrado que la utilización del algoritmo en técnicas de control es factible, optimizando el coste para estos problemas. Por tanto, se puede aplicar el RRA a diversas posibilidades en el mundo de la ingeniería de control. En

este proyecto no se han utilizado poblaciones muy grandes ni un número de iteraciones alto debido a la potencia de computación del ordenador utilizado, pero podrían haberse obtenido resultados incluso mayores al explorar poblaciones más amplias durante más iteraciones.

Además una ventaja de este tipo de algoritmos es su capacidad de escapar de mínimos locales, puesto que cuando no encuentra mejora en un posible mínimo local, el algoritmo se resetea a lo largo del dominio del problema en la búsqueda de otros mínimos. Esto permite la utilización de este algoritmo en una cantidad diversas de problemas.

Asimismo se ha demostrado que es capaz de funcionar en técnicas de control bilineal, lo que abarca aún más su rango de aplicación, desde sistemas lineales de control simples como un regulador de temperatura hasta sistemas más complejos no lineales de control.

Por lo tanto, surgen nuevos trabajos y problemas futuros como posibles aplicaciones de este trabajo, diversos problemas de optimización a la hora de controlar sistemas bilineales ingenieriles, como centrales eléctricas y nucleares, donde se considera el control de las fisiones inducidas por neutrones de isótopos como el uranio o el plutonio, así como los nuevos materiales empleados en la industria aeroespacial cuyas propiedades no son lineales, como el caso de las SMA (Shape Memmory Alloys).

Referencias

[1] J.H. Holland, *«Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology»* Control, and Artificial Intelligence, University of Michigan Press, Oxford, England, 1975.

[2] Minorsky, N. (1922). *«Directional stability of automatically steered bodies»* J. Amer. Soc of Naval Engineers 34: 280-309

[3] R. Medina Garzón y O. Murcia Martínez, 19 de Julio de 2011. *«Antenas Yagi-Uda»*. Disponible en *Slideshare*: https://es.slideshare.net/Arkso/antenas-yagi-uda-8639115

[4] Gregory S. Hornby, Al Globus, Derek S. Linden, Jason D. Lohn (2006). *«Automated Antenna Design with Evolutionary Algorithms».* American Institute of Aeronautics and Astronautics. Disponible en: http://alglobus.net/NASAwork/papers

[5] Dryden Flight Research Center (2002). *«Intelligent Flight Control System»* National Aeronautics and Space Administration. Disponible en: https://www.nasa.gov/centers/dryden/

[6] F. Merrikh-Bayat. *«The runner-root algorithm: A metaheuristic for solving unimodal and multimodal optimization problems inspired by runners and roots of plants in nature».* Applied Soft Computing 33 (2015) 292–303

[7] *«Pelo radical»* Wikipedia, la enciclopedia libre. 10-Mayo-2017.

[8] F. Merrikh-Bayat. 4-Octubre-2014 *«A Numerical Optimization Algorithm Inspired by the Strawberry Plant»*.

[9] *«Heurística (informática)»* Wikipedia, la enciclopedia libre. 15-Mayo-2017

[10] *«Metaheurística»* Wikipedia, la enciclopedia libre. 15-Mayo-2017

[11] Sadiq, S. M. y Habib, Y. (1999). *«Iterative Computer Algorithms with Applications in Engineering»* Solving Combinatorial Optimization Problems.

[12] Goldberg, D. (1989) *«Genetics Algorithms in Search, Optimization and Machine Learning»* Addison Wesley.

[13] P. Tolmos Rodriguez-Piñero. *«Introducción a los algoritmos genéticos y sus aplicaciones»* Universitat de València. Disponible en: https://www.uv.es/asepuma/

[14] F. Sancho Caparrini. *«Algoritmos Genéticos y Computación Evolutiva»* Dpto. de Ciencias de la Computación e Inteligencia Artificial. Universidad de Sevilla. Disponible en: http://www.cs.us.es/~fsancho/

[15] A. García Sánchez. *«Técnicas metaheurísticas»* Escuela Técnica Superior de Ingenieros Industriales. Universidad Politécnica de Madrid. Disponible en: http://www.iol.etsii.upm.es/arch/

[16] J. Arranz de la Peña, A. Parra Truyol. «Algoritmos genéticos» Universidad Carlos III de Madrid. Disponible en: http://www.it.uc3m.es/jvillena/irc/practicas/06-07/05.pdf

[17] Pham D.T. y Karaboga D. (2000). *«Inteligent Optimisation Techniques»* Springer. London

[18] Aniruddha Bhattacharya, Member, IEEE, and Pranab Kumar Chattopadhyay. «Hybrid Differential Evolution With Biogeography-Based Optimization for Solution of Economic Load Dispatch» IEEE transactions on power systems, Vol. 25, No. 4, November 2010

[19] W. Gong, Z. Cai, and C. X. Ling, *«DE/BBO: A Hybrid Differential Evolution With Biogeography-Based Optimization for Global Numerical Optimization»* Disponible en: http://embeddedlab.csuohio.edu/BBO/

[20] «Controlador PID» Wikipedia, la enciclopedia libre. 21-Mayo-2017

[21] A. Flores, D. Copaci, A. Martín, D. Blanco, L. Moreno, *«Smooth and accurate control of multiple shape memory alloys based actuators via low cost embedded hardware»* in: Workshop on Smart Materials and Alternative Techonologies for Bio-inspired Robots and Systems at: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2012.

[22] A. Villoslada, N. Escudero, F. Martín, A. Flores, C. Rivera, M. Collado, L. Moreno. *«Position control of a shape memory alloy actuator using a four-term bilinear PID controller»* Carlos III University, Madrid, Spain y ARQUIMEA Ingeniería S.L.U., Spain

[23] Mohler, R. R. (1973). «Bilinear control processes, mathematics in science and engineering» New York: Academic Press.

[24] F. Martín, C. A. Monje, L. Moreno, C. Balaguer (2015). *«DE-based tuning of $PI^{\lambda}D^{\mu}$ controllers»* Robotics Lab, Department of Systems Engineering and Automation, Carlos III University, Madrid, Spain

[25] Chen YQ, Moore KL. *«Relay feedback tuning of robust PID controllers with iso- damping property»*. IEEE Trans Syst Man Cybern Part B 2005; 35:23–31.

[26] C.A. Coello Coello, *«Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art»* Comput. Methods Appl. Mech. Eng. 191 (11–12) (2002) 1245–1287.

[27] Narendra, K. S. (1996). *«Neural networks for control: Theory and practice»* Proceedings of the IEEE, 84(10), 1385–1406.

[28] Mohler, R. R., & Shen, C. N. (1970). *«Optimal control of nuclear reactors»* New York: Academic Press.

[29] Bruni, C., Dipillo, G., & Koch, G. (1974). *«Bilinear systems: An appealing class of nearly linear systems in theory andapplications»* IEEE Transactions on Automatic and Control, 19, 334–348.

[30] Dunoyer, A. P. (1996). *«Bilinear self-tuning control and bilinearisation of nonlinear industrial systems»* Ph.D. Thesis. Coventry, UK: Coventry University.

[31] S. Martineau, K.J. Burnham, O.C.L. Haas, G. Andrews, A. Heeley, *«Four-term bilinear PID controller applied to an industrial furnace»* Control Eng. Pract. 12 (2004) 457–464.

[32] Matlab. Disponible en: https://es.mathworks.com/products/matlab.html

[33] Simulink. Disponible en: https://es.mathworks.com/products/simulink.html?s_tid=hp_ff_p_simulink

Anexo

Presupuesto del Proyecto

Para calcular el presupuesto del proyecto hay que tener en cuenta tres factores: las horas de trabajo, el hardware y el software.

La dedicación a este proyecto en cuanto a horas de trabajo también hay que dividirlas en horas de trabajo del alumno y horas de trabajo con el tutor.

Por lo tanto, en el cálculo de las horas trabajadas por el alumno se han dedicado alrededor de 21 semanas, trabajando aproximadamente 18 horas a la semana, obteniendo un resultado total de 378 horas dedicadas al proyecto. Suponiendo que las horas trabajadas por el alumno se remuneran a 10 euros por hora, se obtiene un coste total por horas trabajadas por el alumno de 3780 euros.

En cuanto a las horas trabajadas con el tutor, se han dedicado alrededor de 15 horas de trabajo durante el proyecto. La remuneración de las horas trabajadas con el tutor es mayor que la del alumno, suponiendo a 20 euros por hora, por lo que resulta en un coste de horas trabajadas con el tutor de 300 euros.

Para la realización de este proyecto se ha precisado de una computadora en la que poder realizar las simulaciones, cálculos y la programación. Esto se corresponde con el hardware del proyecto, el cual en concreto es un ordenador Toshiba Satellite L855-11K de 2012 con un valor de 500 euros.

Además, el entorno de programación y simulación de los resultados utilizado es el software Matlab y Simulink R2016b con una licencia de estudiante por valor de 35 euros.

El presupuesto se puede ver reflejado en la Tabla 5.

Presupuesto del proyecto (Euros)				
Horas de trabajo del alumno	Horas de trabajo con el tutor	Hardware	Software	TOTAL
3780	300	500	35	4615

Tabla 5. Presupuesto del proyecto

Función del algoritmo RRA implementada en Matlab


```
%Marcos Marín Rodríguez
%Runner Root Algorithm (RRA)

%   - Description:  This module optimizes the parameters of a Bilinear
%       PID in order to control a given plant (defined below by Gp).
%       The Bilinear PID Controller is defined by the following expression:
%       Blinear Term: u(k)=(1+Kb*yref(k))/(1+Kb*y(k-1))*v(k)
%       PID(s)=Kp + Ki/s + Kd*s

%-------------------------------------------------------------------------
%   - Initialization Parameters:
%-------------------------------------------------------------------------
%   - Options:
%       - Cost function:
%           1: time-domain, squared error of the step response
%           2: frequency-domain, plane argument in the bode diagram about
%           the phase margin frequency

cost_function=1;

%-------------------------------------------------------------------------
%   - Transfer function of the process to be controlled.
Jj = 0.01;
b = 0.1;
Kk = 0.01;
R = 1;
L = 0.5;
%s = tf('s');
%Gp = Kk/(s*((Jj*s+b)*(L*s+R)+Kk^2));
Gp=tf(0.01,[0.005 0.06 0.1001]);

%-------------------------------------------------------------------------
%   - Maximum values of BPID constants

MAX_VALUE_KP = 1;
MAX_VALUE_KI = 1;
MAX_VALUE_KD = 1;
MAX_VALUE_KB = 1;

%-------------------------------------------------------------------------
%   - RRA variables
Npop=5;         %Population size
stall_max=10;   %Maximum iteration number
tol=0.01;       %Tolerance
a=0.1;          %Selection intensity value
CROMOSOMAS=4;   %Number of cromosomes

%   - Upper limit vector
Xu = [MAX_VALUE_KP,MAX_VALUE_KI,MAX_VALUE_KD,MAX_VALUE_KB];
%   - Lower limit
Xl=0;

%   - Runner and root lengths
d_runner=Xu*0.5;
d_root=Xu*0.01;
```

```
X_mother=zeros(Npop, CROMOSOMAS+2);
X_daughter=zeros(Npop, CROMOSOMAS+2);
X_perturbed=zeros(CROMOSOMAS, CROMOSOMAS+2);

%-------------------------------------------------------------------------
%   - The initial population is generated
for k=1:Npop
    for j=2:CROMOSOMAS
    X_mother(k,j)=X1+rand*(Xu(j-1)-X1);
    end
end

%   - Cost function evaluation for the initial population
for k=1:Npop
    X_mother(k,1)=cost(Gp,X_mother(k,2:(CROMOSOMAS+1)),cost_function);
end

%   - Arranging population by cost value
AUX=sortrows(X_mother,1);
X_mother=AUX;

stall_count=0;

%   - Global search procedure
for i=1:stall_max
    X_daughter(1,:)=X_mother(1,:); %First daughter equal to best mother

    for k=2:Npop
        r=unifrnd(-0.5,0.5,CROMOSOMAS,1);

        %   - Mutation process
        for j=2:CROMOSOMAS+1
            X_daughter(k,j)= X_mother(k,j) + d_runner(j-1)*r(j-1);
            if X_daughter(k,j) < 0
                X_daughter(k,j)=0.001;
            end
            if X_daughter(k,j) > Xu(j-1)
                X_daughter(k,j)=Xu(j-1);

            end
        end

    end


%   - Cost function evaluation for the new population
    for k=1:Npop
        X_daughter(k,1)=cost(Gp,X_daughter(k,2:(CROMOSOMAS+1)),cost_function);
    end

%   - Arranging population by cost value
    AUX=sortrows(X_daughter,1);
    X_daughter=AUX;
```

```
    toler=(min(X_daughter(:,1))-min(X_mother(:,1)))/min(X_mother(:,1));
    lugar_minimo=1;

    %   - Local search procedure
    if i>1 && abs(toler)<tol
        fprintf(1,'\nBúsqueda local...')

        %   - Big steps local search
        fprintf(1,' - Amplios pasos')
        for k=1:CROMOSOMAS
            Nk=rand-0.5;
            X_perturbed(k,:)=X_daughter(lugar_minimo,:);
            X_perturbed(k,k+1)=X_perturbed(k,k+1)+d_runner(k)*Nk;

            if X_perturbed(k,k+1) < 0
                X_perturbed(k,k+1)=0.001;
            end

            if X_perturbed(k,k+1) > Xu(k)
                    X_perturbed(k,k+1)=Xu(k);
            end

            %   - Cost function evaluation for the new plant
            X_perturbed(k,1)=cost(Gp,X_perturbed(k,2:(CROMOSOMAS+1)),cost_function);

            if X_perturbed(k,1)<X_daughter(lugar_minimo,1)
                X_daughter(lugar_minimo,:)=X_perturbed(k,:);
            end

        end

        %   - Small steps local search
        fprintf(1,' - Pequeños pasos')
        for k=1:CROMOSOMAS
            Rk=unifrnd(-0.5,0.5);
            X_perturbed(k,:)=X_daughter(lugar_minimo,:);
            X_perturbed(k,k+1)=X_perturbed(k,k+1)+d_root(k)*Rk;
            if X_perturbed(k,k+1) < 0
                X_perturbed(k,k+1)=0.001;
            end
            if X_perturbed(k,k+1) > Xu(k)
                    X_perturbed(k,k+1)= Xu(k);
            end

            %   - Cost function evaluation for the new plant
            X_perturbed(k,1)=cost(Gp,X_perturbed(k,2:(CROMOSOMAS+1)),cost_function);

            if X_perturbed(k,1)<X_daughter(lugar_minimo,1)
                X_daughter(lugar_minimo,:)=X_perturbed(k,:);
            end
        end
    end

%   - Stall condition (local minimum point)
```

```
    toler=(min(X_daughter(:,1))-min(X_mother(:,1)))/min(X_mother(:,1));
    if i>1 && abs(toler)<tol
        stall_count=stall_count+1;
    else
        stall_count=0;
    end
%   - Second evaluation with no better results
    if stall_count == 2
        X_mother(1,:)=X_daughter(lugar_minimo,:);
        %   - Reset population but best member
        for k=2:Npop
            for j=2:CROMOSOMAS
                X_mother(k,j)=Xl+rand*(Xu(j-1)-Xl);
            end
        end
        stall_count=0;
    else

%   - Selection process
        %   - First mother equal to best daughter (Elite selection)
        X_mother(1,:)=X_daughter(lugar_minimo,:);

        %   - Roulette wheel selection
        for k=1:Npop
            fitness(k)=1/(a+X_daughter(k,1)-X_daughter(lugar_minimo,1));
        end

        for k=1:Npop
            Prob=fitness(k)/sum(fitness);
            X_daughter(k,CROMOSOMAS+2)=Prob;
        end

        for k=2:Npop
            chosen_index = fortune_wheel(X_daughter(:,CROMOSOMAS+2));
            X_mother(k,:) = X_daughter(chosen_index,:);
        end
    end

%   - Showing results
        X_mother(1,1)=cost(Gp,X_mother(1,2:(CROMOSOMAS+1)),cost_function);
        fprintf(1,'\n It: %f Best %f \n Parameters: Kp: %f Ki: %f Kd: %f Kb: %f \n',i,↙
X_mother(1,1),X_mother(1,2),X_mother(1,3),X_mother(1,4),X_mother(1,5));

end
```

Función de selección por rueda de ruleta

21/06/17 13:27 E:\RRA\Matlab\fortune_wheel.m 1 of 1

```
% ------------------------------------------------------------
% Roulette Wheel Selection Algorithm. A set of weights
% represents the probability of selection of each
% individual in a group of choices. It returns the index
% of the chosen individual.
% Usage example:
% fortune_wheel ([1 5 3 15 8 1])
% most probable result is 4 (weights 15)
% ------------------------------------------------------------
function choice = fortune_wheel(weights)
accumulation = cumsum(weights);
p = rand() * accumulation(end);
chosen_index = -1;

for index = 1 : length(accumulation)

if (accumulation(index) > p)
chosen_index = index;
break;

end
end

choice = chosen_index;

end
```

Función de coste utilizada para el algoritmo RRA

21/06/17 13:38 E:\RRA\4 Control_BPID_V0\Contro...\cost.m 1 of 3

```
function[error]=cost(Gp,param,option)
%-------------------------------------------------------------------------
% Function: Cost Function.
% Author: Fernando Martin Monar.
% Date: November, 2015
%-------------------------------------------------------------------------
% -> Description:
% It calculates the cost function given a process Gp and a PID
controller.
% The DE-based scan matching algorithm calls it to estimate the cost.
% The Bilinear PID Controller is defined by the following expression:
% Blinear Term: u(k)=(1+Kb*yref(k))/(1+Kb*y(k-1))*v(k)
% PID(s)=Kp + Ki/s + Kd*s
% Several options have been defined for the cost function:
% 1: Time-domain, squared error of the step response. This
% version runs simulink in order to obtain the error, taking
% advantage of the saturation blocks that can be used to limit
% the control signal. However, Simulink works slowly.
%-------------------------------------------------------------------------
% -> Usage:
% Inputs: Gp: transfer function of the process to control
% param = [Kp Ki Kd Kb]; parameters of the BPID
% option: cost function.
% Output: error.
%-------------------------------------------------------------------------
% -> See also: Control_BPID nid
%-------------------------------------------------------------------------
% Initialization parameters.

warning('OFF');
PHASE=zeros(11,1);
fr_w=0.2;

if option==1
      %OPTION 1: step response
      Kp=param(1);
      Ki=param(2);
      Kd=param(3);
      Kb=param(4);

      assignin('base', 'Kp', Kp );
      assignin('base', 'Ki', Ki );
      assignin('base', 'Kd', Kd );
      assignin('base', 'Kb', Kb );

      [numG, denG] = tfdata(Gp);
      assignin('base', 'numG', numG );
      assignin('base', 'denG', denG );
      numMatG = cell2mat(numG);
      denMatG = cell2mat(denG);
      assignin('base', 'NumGp', numMatG );
      assignin('base', 'DenGp', denMatG );

      sim('SIM_MODEL'); %Simulacion modelo
      s=size(simout.signals.values);
```

```
      m = s(1,1); %Numero de muestras
      t = simout.signals.values(1:m,1); %Tiempo
      ErrInst = simout.signals.values(1:m,2); %Error InstantÃ¡neo
      d = simout.signals.values(1:m,3); %Respuesta
      TempFin = t(m,1);
      Maximo = max(d);
      assignin('base', 's', s);
      assignin('base', 'm', m);
      assignin('base', 't', t);
      assignin('base', 'd', d);
      assignin('base', 'ErrInst', ErrInst);
      assignin('base', 'TempFin', TempFin);
      assignin('base', 'Maximo', Maximo);

      ErrInt = sum(abs(ErrInst));
      error = ErrInt/m;
      assignin('base', 'error', error);

%without simulink
      %M=feedback(PID*Gp,1);
      %[Y]=step(M);
      %if Y(30)<0.8
      % error=(Y-1)'*(Y-1); % The squared error between the output and
      a unitary
      % step is computed
      %error=(max(Y)-1)/1;
      %else
      % error=10000000;
      %end

end

end
```